Undecidability of Universality for Timed Automata with Minimal Resources

Sara Adams, Joël Ouaknine, and James Worrell

Oxford University Computing Laboratory
{sara.adams,joel,jbw}@comlab.ox.ac.uk

Abstract. Timed automata were introduced by Alur and Dill in the early 1990s and have since become the most prominent modelling formalism for real-time systems. A fundamental limit to the algorithmic analysis of timed automata, however, results from the undecidability of the universality problem: does a given timed automaton accept every timed word? As a result, much research has focussed on attempting to circumvent this difficulty, often by restricting the class of automata under consideration, or by altering their semantics.
In this paper, we study the decidability of universality for classes of timed automata with minimal resources. More precisely, we consider restrictions on the number of states and clock constants, as well as the size of the event alphabet. Our main result is that universality remains undecidable for timed automata with a single state, over a single-event alphabet, and using no more than three distinct clock constants.

1 Introduction

Timed automata were introduced by Alur and Dill in [3] as a natural and versatile model for real-time systems. They have been widely studied ever since, both by practitioners and theoreticians. A celebrated result concerning timed automata, which originally appeared in [2] in a slightly different context, is the PSPACE decidability of the *language emptiness* (or *reachability*) problem.

Unfortunately, the *language inclusion* problem—given two timed automata $\mathcal{A}$ and $\mathcal{B}$, is every timed word accepted by $\mathcal{A}$ also accepted by $\mathcal{B}$?—is known to be undecidable. This severely restricts the algorithmic analysis of timed automata, both from a practical and theoretical perspective, as many interesting questions can be phrased in terms of language inclusion. Over the past decade, several researchers have therefore attempted to circumvent this negative result by investigating language inclusion, or closely related concepts, under various assumptions and restrictions. Among others, we note the use of (i) topological restrictions and digitization techniques: [10,6,18,15,17]; (ii) fuzzy semantics: [9,11,16,5]; (iii) determinisable subclasses of timed automata: [4,20]; (iv) timed simulation relations and homomorphisms: [21,13,12]; and (v) restrictions on the number of clocks: [19,7].

The undecidability of language inclusion, first established in [3], derives from the undecidability of an even more fundamental problem, that of *universality*:

does a given timed automaton accept every timed word? Research has shown the undecidability of universality to be quite robust, although it does break down under certain (fairly stringent) hypotheses, which we survey in Section 3.

The goal of the present paper is to study the (un)decidability of universality for classes of timed automata with *minimal resources*. Here 'resource' refers to quantities such as number of discrete states, number of clocks, size of the alphabet, number or magnitude of clock constants, etc. Our main result is that the universality problem remains undecidable for timed automata with a single state, over a single-event alphabet, and using the clock constants 0 and 1 only.[1]

At a conceptual level, one can paraphrase this result as asserting the existence of an undecidable problem for *stateless* and *eventless* real-time systems. Any computation carried out by such a device must rely exclusively on clocks, which can be viewed as some kind of unwieldy 'analog' memory. Clocks can only be reset (to zero) and compared against the constants 0 and 1;[2] moreover, they continually increase with time, and hence are quite poor at holding information over any given period of time.

One potential application of our work lies in establishing further decision problems about real-time systems to be undecidable; in this respect, the absence of any state or event structure in the formulation of our problem suggests it should be a well-suited target for a wide variety of real-time formalisms.

This paper summarises the key ideas and constructions of [1], to which we refer the reader for full details.

2 Timed Automata

Let X be a finite set of clocks, denoted x, y, z, etc. We define the set Φ_X of clock constraints over X via the following grammar, where $k \in \mathbb{N}$ stands for any non-negative integer, and $\bowtie \in \{=, \neq, <, >, \leq, \geq\}$ is a comparison operator:

$$\phi ::= \mathbf{true} \;|\; x \bowtie k \;|\; \phi \wedge \phi \;|\; \phi \vee \phi.$$

A *timed automaton* is a six-tuple $(\Sigma, S, S_0, S_f, X, \Delta)$, where

- Σ is a finite set (alphabet) of events,
- S is a finite set of states,
- $S_0 \subseteq S$ is a set of start states,
- $S_f \subseteq S$ is a set of accepting states,
- X is a finite set of clocks, and
- $\Delta \subseteq S \times S \times \Sigma \times \Phi_X \times \mathcal{P}(X)$ is a finite set of transitions. A transition (s, s', a, ϕ, R) allows a jump from state s to s', consuming event $a \in \Sigma$ in the process, provided the constraint ϕ on clocks is met. Afterwards, the clocks in R are reset to zero, while all other clocks remain unchanged.

[1] This result holds over *weakly monotonic* time; over *strongly monotonic* time, we require the clock constants 1, 2, and 3 instead. Full details are presented in Section 3.

[2] Note that we do not allow comparing clocks against each other.

Remark 1. An important observation is that *diagonal* clock constraints (of the form $x - y \bowtie k$) are not allowed in our model of timed automata. This restriction strengthens our main single-state undecidability results, in that multiple states cannot simply be encoded through the fractional ordering of clock values; see [22].

We will abuse notation and allow transitions to be labelled by *sets* of events, in addition to individual events. A transition labelled by $U \subseteq \Sigma$ simply corresponds to $|U|$ copies of the transition, one for each event in U.

For the remainder of this section, we assume a fixed timed automaton $\mathcal{A} = (\Sigma, S, S_0, S_f, X, \Delta)$.

A *clock valuation* is a function $\nu : X \to \mathbb{R}^+$, where $\mathbb{R}^+$ stands for the non-negative real numbers. If $t \in \mathbb{R}^+$, we let $\nu + t$ be the clock valuation such that $(\nu + t)(x) = \nu(x) + t$ for all $x \in X$.

A *configuration* of $\mathcal{A}$ is a pair (s, ν), where $s \in S$ is a state and ν is a clock valuation.

An *accepting run* of $\mathcal{A}$ is a finite alternating sequence of configurations and delayed transitions $\pi = (s_0, \nu_0) \xrightarrow{d_1,\theta_1} (s_1, \nu_1) \xrightarrow{d_2,\theta_2} \ldots \xrightarrow{d_n,\theta_n} (s_n, \nu_n)$, where $d_i \in \mathbb{R}^+$ and $\theta_i = (s_{i-1}, s_i, a_i, \phi_i, R_i) \in \Delta$, subject to the following conditions:

1. $s_0 \in S_0$, and for all $x \in X$, $\nu_0(x) = 0$.
2. For all $0 \leq i \leq n-1$, $\nu_i + d_{i+1}$ satisfies ϕ_{i+1}.
3. For all $0 \leq i \leq n-1$, $\nu_{i+1}(x) = \nu_i(x) + d_{i+1}$ for all $x \in X \setminus R_{i+1}$, and $\nu_{i+1}(x) = 0$ for all $x \in R_{i+1}$.
4. $s_n \in S_f$.

Each d_i is interpreted as the time delay between the firing of transitions, and each configuration (s_i, ν_i), for $i \geq 1$, records the data immediately following transition θ_i. Abusing notation, we also write runs in the form $(s_0, \nu_0) \xrightarrow{d_1,a_1} (s_1, \nu_1) \xrightarrow{d_2,a_2} \ldots \xrightarrow{d_n,a_n} (s_n, \nu_n)$ to highlight the run's events.

A *timed word* is a pair (σ, τ), where $\sigma = \langle a_1 a_2 \ldots a_n \rangle \in \Sigma^*$ is a word and $\tau = \langle t_1 t_2 \ldots t_n \rangle \in (\mathbb{R}^+)^*$ is a non-decreasing sequence of real-valued timestamps of the same length.

Such a timed word is *accepted* by $\mathcal{A}$ if $\mathcal{A}$ has some accepting run of the form $\pi = (s_0, \nu_0) \xrightarrow{d_1,a_1} (s_1, \nu_1) \xrightarrow{d_2,a_2} \ldots \xrightarrow{d_n,a_n} (s_n, \nu_n)$ where, for each $1 \leq i \leq n$, $t_i = d_1 + d_2 + \ldots + d_i$.

Remark 2. Our timed semantics is *weakly monotonic*, in that it allows multiple events to occur 'simultaneously', or, more precisely, with null-duration delays between them. We will also consider an alternative semantics, termed *strongly monotonic*, which requires the timestamps in timed words to be strictly increasing. As it turns out, our main undecidability results remain essentially the same, although some of the constructions have to be altered in places.

3 The Universality Problem

Consider a class of computational machines that act as language acceptors, such as finite automata or pushdown automata. A particular machine is said to be *universal* if it accepts all possible words (over the relevant alphabet). The *universality problem*, for the class of machines in question, consists in determining whether a given machine is universal or not.

The universality problem is a cornerstone of formal language theory, and has been extensively studied in a wide array of contexts. Moreover, it is the simplest instance of the language inclusion problem, since virtually all interesting computational classes will comprise universal machines.

In this paper, we are interested in universality for certain restricted classes of timed automata. This problem splits naturally into two main instances, according to the assumption made on the monotonicity of time. A timed automaton is said to be *universal over weakly monotonic time* if it accepts all timed words (over its alphabet), and is said to be *universal over strongly monotonic time* if it accepts all timed words in which the timestamps are strictly increasing.

3.1 Background

The most fundamental result concerning universality for timed automata is Alur and Dill's proof of undecidability in the general case (over both weakly and strongly monotonic time) [3]. An examination of that proof reveals that universality is in fact undecidable for the class of timed automata having at most two clocks.

Much more recently, it was discovered that universality is decidable for timed automata having at most one clock (irrespective of the monotonicity of time) [19], using results from the theory of well-structured transition systems [8]. Together with Alur and Dill's work, this completely classifies the decidability of universality as a function of the number of clocks allowed.

The paper [19] also proves that universality is decidable for timed automata that only make use of the clock constant 0 in clock constraints. On the other hand, [3] shows that allowing any additional clock constant leads to undecidability.

Henzinger et al. introduced the notion of digitization in [10]. Using this technique, it is possible to show that, over weakly monotonic time, universality is decidable for open[3] timed automata. Universality is however undecidable for closed timed automata (regardless of the monotonicity of time) as well as for open timed automata over strongly monotonic time [18].

3.2 Main Results

The primary focus of this paper is to study universality for timed automata in which the number of states, the size of the alphabet, and the number of different clock constants are simultaneously restricted. Our main results are as follows:

[3] A timed automaton is open if it only uses open (strict) comparison operators in clock constraints, i.e., $\{<, >, \neq\}$.

Theorem 1. *Over weakly monotonic time, the universality problem is undecidable for timed automata with a single state, a single-event alphabet, and using clock constants* 0 *and* 1 *only.*

Theorem 2. *Over strongly monotonic time, the universality problem is undecidable for timed automata with a single state, a single-event alphabet, and using clock constants* 1*,* 2*, and* 3 *only.*

Remark 3. The above restrictions (number of states, size of alphabet, and number of clock constants) seemed to us the most natural and important ones to consider. We note that there would be no point in restricting in addition the number of clocks, since the total number of instances of such timed automata would then essentially be finite, vacuously making any decision problem decidable.

We also note that bounding the number of transitions, even on its own, would likewise result in a finite number of 'distinguishable' timed automata, and would therefore also make decidability issues moot.

The next section is devoted to proving Theorem 1. A proof sketch of Theorem 2 is given in Section 5.

4 Universality over Weakly Monotonic Time

At a high level, Alur and Dill's original undecidability proof runs as follows. Take a two-counter machine $\mathcal{M}$ with a distinguished halting state, and produce an encoding of its runs as timed words. In this encoding, a step or instruction of $\mathcal{M}$ is carried out every time unit; the values of the counters are encoded as repeated, non-simultaneous events, by exploiting the density of the real numbers to accommodate arbitrarily large numbers. One then manufactures a timed automaton $\mathcal{A}$ that accepts all timed words that do not correspond to some encoding of a valid halting run of $\mathcal{M}$. In other words, $\mathcal{A}$ is universal iff $\mathcal{M}$ does not halt, which immediately entails the undecidability of universality.

The construction of $\mathcal{A}$ makes heavy use of nondeterminism. More precisely, $\mathcal{A}$ accepts the encodings of all 'runs' that are either invalid or non-halting. The latter is easy to detect: a run is non-halting if it doesn't end in the distinguished halting state of $\mathcal{M}$. Invalid runs, on the other hand, exhibit at least one local violation, e.g., the consecutive values of one of the counters are inconsistent at some stage, or an illegal jump was made from one state to another, or the timed word does not respect the prescribed format, etc. In each case, $\mathcal{A}$ uses nondeterminism to 'find' and expose the local violation, and accept the corresponding timed word.

Our task here is to reproduce this overall approach under the stringent limitations of Theorem 1. The main ingredients are as follows:

1. We show that one can construct $\mathcal{A}$ to be a linear safety timed automaton, i.e., an automaton whose only loops are self-loops and all of whose states are accepting.

2. Moreover, we can reduce the alphabet of $\mathcal{A}$ to a single letter by encoding different symbols as fixed numbers of 'simultaneous' events.
3. Since $\mathcal{A}$ is linear, it can only change control states a bounded number of times over any run. Assume that all clocks initially have value greater than or equal to 1 (which can be achieved by accepting any behaviour during the first time unit). We can then reduce $\mathcal{A}$ to having a single state, by encoding other states through various combinations of certain clocks having value less than 1 and other clocks having value greater than or equal to 1.
4. Finally, the overall correctness of the construction requires that various technical conditions and invariants in addition be maintained throughout runs of $\mathcal{A}$; for example, if the delay between two consecutive events ever exceeds 1, then all subsequent behaviours should be accepted.

We now present the technical arguments in detail.

4.1 Two-Counter Machines

A *two-counter machine* $\mathcal{M}$ is a six-tuple (Q, q_0, q_f, C, D, Ξ), where Q is a finite set of states, $q_0 \in Q$ is the initial state, $q_f \in Q$ is the halting state, C and D are two counters ranging over the non-negative integers, and Ξ is the transition relation. Both counters are initially empty, and $\mathcal{M}$ starts in state q_0. Every state except q_f has a unique outgoing transition in Ξ associated to it. Such a transition can either: (i) increment or decrement (if non-zero) one of the counters, and subsequently jump to a new state; or (ii) test one of the counters for emptiness and conditionally jump to a new state.

A *configuration* of $\mathcal{M}$ is a triple (q, c, d), where q is the current state and c, d are the respective values of the counters. A configuration is *halting* if it is of the form (q_f, c, d). We assume that the transition relation is fully deterministic, so that each non-halting configuration has a unique successor. It is well-known that the halting problem for two-counter machines, i.e., whether a halting configuration is reachable from $(q_0, 0, 0)$, is undecidable [14].

We associate to any given two-counter machine $\mathcal{M}$ a set of strongly monotonic timed words $L(\mathcal{M})$ that are encodings of the halting computations of $\mathcal{M}$. Our alphabet is $\Sigma = Q \cup \{a, b\}$ (where we assume $a, b \notin Q$). If $\mathcal{M}$ does not halt, then naturally we let $L(\mathcal{M}) = \emptyset$. We otherwise note that, since $\mathcal{M}$ is deterministic, it has at most one valid halting computation, which we denote $\pi = \langle (s_1, c_1, d_1)(s_2, c_2, d_2), \ldots, (s_n, c_n, d_n) \rangle$. We then include in $L(\mathcal{M})$ all timed words (σ, τ) that satisfy the following:

1. $\sigma = \sigma_{\mathrm{init}} \sigma' \sigma_{\mathrm{end}}$, where $\sigma' = (s_1 a^{c_1} b a^{d_1})(s_2 a^{c_2} b a^{d_2}) \ldots (s_{n-1} a^{c_{n-1}} b a^{d_{n-1}}) s_n$, and $\sigma_{\mathrm{init}}, \sigma_{\mathrm{end}} \subseteq \Sigma^*$. In other words, σ can be decomposed as a prefix σ_{init}, in which anything is allowed, followed by σ', which is a fairly straightforward encoding of π, and capped by a suffix σ_{end}, in which anything is allowed once again.[4] The values of the counters in configurations correspond to the

[4] As we will aim to capture the complement of $L(\mathcal{M})$ using a safety automaton—whose language is therefore prefix closed—it is necessary for $L(\mathcal{M})$ to be suffix closed.

number of consecutive a's, with b acting as a delimiter between the encodings of the first and the second counter.
2. τ is strongly monotonic.
3. $\tau = \tau_{\text{init}}\tau'\tau_{\text{end}}$, where τ_{init}, τ', and τ_{end} are sequences whose lengths respectively match those of σ_{init}, σ', and σ_{end}. Moreover, all timestamps in τ_{init} are strictly less than 1.
4. The associated timestamp of each s_i in σ' is i. Moreover, if the timestamp of the first b in σ' is $1 + t_b$, then the timestamp of the ith b is $i + t_b$. An immediate consequence is that the time delay between successive events in σ' is always strictly less than 1.
5. For all $1 \leq i \leq n-2$: (i) if $c_{i+1} = c_i$, then for each a with timestamp t in the time interval $(i, i+t_b)$, there is an a with timestamp $t+1$ in the time interval $(i+1, i+1+t_b)$; (ii) if $c_{i+1} = c_i + 1$, then for each a with timestamp $t+1$ in the interval $(i+1, i+1+t_b)$, except the last one, there is an a with timestamp t in the interval $(i, i+t_b)$; (iii) if $c_{i+1} = c_i - 1$, then for each a with timestamp t in the interval $(i, i+t_b)$, except the last one, there is an a with timestamp $t+1$ in the interval $(i+1, i+1+t_b)$; (iv) similar requirements hold for the second counter.

By construction, $\mathcal{M}$ halts iff $L(\mathcal{M})$ is not empty.

4.2 Linear Safety Timed Automata

Given a two-counter machine $\mathcal{M} = (Q, q_0, q_f, C, D, \Xi)$ as above, we sketch how to construct a timed automaton $\mathcal{A}$ that accepts precisely all strongly monotonic timed words that do not belong to $L(\mathcal{M})$. We ensure that $\mathcal{A}$ enjoys a number of technical properties, as follows:

1. $\mathcal{A}$ is *linear*, i.e., the only loops in its transition relation are self-loops. As a result, $\mathcal{A}$ can only change control states a bounded number of times over any run.
2. $\mathcal{A}$ is a *safety* timed automaton, i.e., all of its states are accepting.
3. As per the definition of $L(\mathcal{M})$, $\mathcal{A}$'s alphabet is $\Sigma = Q \cup \{a, b\}$.
4. $\mathcal{A}$ has a unique state, which we call the *sink* state, from which there is no transition to a different state. $\mathcal{A}$ moves into the sink state as soon as a violation is detected; we therefore postulate a Σ-labelled self-loop on the sink state, i.e., it accepts any behaviour.
5. $\mathcal{A}$ makes use of several clocks, some of which ensure that $\mathcal{A}$ only accepts strongly monotonic timed words.[5]
   Two important clocks are *abs* and *ev*. The clock *abs* is never reset and therefore indicates the total amount of time elapsed since the beginning of a run. After time 1, the clock *ev* is meant to be reset whenever an event occurs, unless a violation has already been detected. More precisely, *ev* obeys the following rules:

[5] Although the aim of the present section is to establish an undecidability result over *weakly* monotonic timed words, the automata we are considering here, which act as intermediate tools, are required to accept only *strongly* monotonic timed words.

- ev is never reset while $abs < 1$.
- ev is not reset on transitions leading to the sink state; in particular, the sink state itself never resets ev.
- From every state, there is a Σ-labelled transition to the sink state guarded by the clock constraint $(abs > 1 \wedge ev \geq 1)$. In other words, once the absolute time exceeds 1, we require that events always occur less than 1 time unit apart, otherwise we record a violation.
- ev is always reset when $(abs > 1 \wedge ev < 1)$, and when $(abs = 1 \wedge ev = 1)$, unless the transition is headed to the sink state.

6. $\mathcal{A}$ accepts any behaviour before time 1; this is achieved by postulating a Σ-labelled self-loop on initial states, guarded by $abs < 1$.
7. $\mathcal{A}$ cannot leave an initial state until at least time 1; this is achieved by guarding all transitions leading away from an initial state by $abs \geq 1$.
8. Let us call an *inner* state one that is neither an initial state nor the sink state. If an inner state has a self-loop, it must be be labelled either by a or by $\Sigma \setminus \{q_f\}$.
9. After time 1, q_f can only label transitions that are headed into the sink state, i.e., once a violation has already been detected; otherwise, $\mathcal{A}$ would potentially accept encodings of valid halting computations of $\mathcal{M}$.

It remains to show that we can indeed construct a timed automaton $\mathcal{A}$ that captures the complement of $L(\mathcal{M})$ and has Properties 1–9. This is achieved as in Alur and Dill's proof, by amalgamating various simple automata, each of which captures some local violation of a halting computation of $\mathcal{M}$.[6] Suppose, for example, that from state q_3, $\mathcal{M}$ is supposed to increment counter D, and move to state q_5. One possible violation of this transition could consist in failing to increment D as per the encoding prescribed in Subsection 4.1, which requires the insertion of an extra a before q_5 in the corresponding timed word. Figure 1 depicts an automaton that captures precisely this behaviour in the case D is originally non-empty.

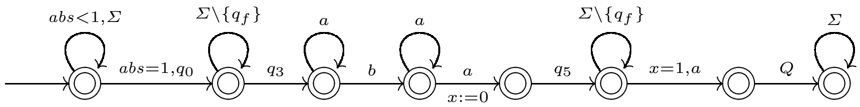


Fig. 1. A linear safety timed automaton that captures an incrementation violation on the second counter. In the interest of clarity, we have omitted data pertaining to Property 5 (strong monotonicity and the treatment of clock ev).

The (numerous) other cases are equally straightforward; full details can be found in [1].

[6] Note that our automata are trivially closed under union.

4.3 Restricting to a Single Event

Let $\mathcal{M} = (Q, q_0, q_f, C, D, \Xi)$ be a two-counter machine. We give a way to encode, or 'flatten', any strongly monotonic timed language L over alphabet $\Sigma = Q \cup \{a, b\}$ into a weakly monotonic timed language $\widetilde{L}$ over the singleton alphabet $\{a\}$.

We define a renaming relation $\mathbf{R}$ from Σ to (untimed) words over alphabet $\{a\}$. This relation will be one-to-one for all events in Σ apart from q_f.

We first set $a\mathbf{R}a$ and $b\mathbf{R}aa$, i.e., the letter 'a' is renamed to the word 'a', and the letter 'b' is renamed to the word 'aa'. Next, let $s_1, \ldots, s_m$ be an enumeration of $Q \setminus \{q_f\}$. We set $s_i\mathbf{R}a^{i+2}$, for $1 \leq i \leq m$. Finally, for all $j \geq 1$, we set $q_f\mathbf{R}a^{m+2+j}$. In other words, q_f renames to the words a^{m+3}, a^{m+4}, a^{m+5}, $\ldots$.

Note that the renaming relation is total and surjective, in that every event in Σ renames to one or more words over $\{a\}$, and every non-empty word over $\{a\}$ is the renaming of some event in Σ.

The renaming relation $\mathbf{R}$ lifts naturally to a renaming relation from strongly monotonic timed words over Σ to weakly monotonic timed words over $\{a\}$. Formally, write $u\mathbf{R}\widetilde{u}$ iff, for every event e with timestamp t in u, there are exactly k 'simultaneous' events a with timestamp t in $\widetilde{u}$, for some k such that $e\mathbf{R}a^k$, and vice-versa. For L a strongly monotonic timed language over Σ, we then let $\widetilde{L}$ be the image of L under this renaming. It is immediate that L is Σ-universal over strongly monotonic time iff $\widetilde{L}$ is $\{a\}$-universal over weakly monotonic time.

Let the timed automaton $\mathcal{A}$ be defined as in the previous subsection, i.e., the strongly monotonic language $L(\mathcal{A})$ of $\mathcal{A}$ is precisely the complement of $L(\mathcal{M})$ with respect to strongly monotonic timed words. Our next task is to define a single-event linear safety timed automaton $\widetilde{\mathcal{A}}$ that accepts precisely the language $\widetilde{L(\mathcal{A})}$.

$\widetilde{\mathcal{A}}$ can be obtained from $\mathcal{A}$ in a straightforward manner. Consider first an e-labelled transition of $\mathcal{A}$ that is not a self-loop, with $e \in \Sigma$. If $e \neq q_f$, then replace this transition by a sequence of k instantaneous a-labelled transitions, where $e\mathbf{R}a^k$. (Instantaneity can be enforced via a clock constraint such as $ev = 0$ on transitions.) If $e = q_f$, on the other hand, Property 9 guarantees that the transition is headed to the sink state, from which anything will be accepted, so it likewise suffices to replace the original transition by a sequence of $m+3$ instantaneous a-labelled transitions that end in the sink state.

The case of self-loops is somewhat more subtle. It is clear that we need only handle self-loops on inner states, since any behaviour is allowed in any case in initial states (while $abs < 1$) and in the sink state. Property 8, however, guarantees that self-loops on inner states are either labelled by a or by $\Sigma \setminus \{q_f\}$. In the first case there is clearly nothing to change. In the second case, we assume that $\widetilde{\mathcal{A}}$ has the use of $m+2$ special clocks $y_1, y_2, \ldots, y_{m+2}$. Whenever an event occurs, $\widetilde{\mathcal{A}}$ resets the y-clock of lowest index that is not already zero. If all y-clocks are zero, no transition is enabled. In this way, no more than $m+2$ consecutive 'simultaneous' a's are possible from any inner state, in effect preventing encodings of q_f from occurring. (Note that as soon as some non-null amount of time elapses, all y-clocks automatically have non-zero values again.)

Stringing everything together, we have that $\mathcal{M}$ does not halt iff $\mathcal{A}$ is Σ-universal over strongly monotonic time iff the single-event timed automaton $\widetilde{\mathcal{A}}$ is $\{a\}$-universal over weakly monotonic time. Moreover, it is immediate from our construction that $\widetilde{\mathcal{A}}$ also satisfies Properties 1–9, *mutatis mutandis* (in particular, $\widetilde{\mathcal{A}}$'s alphabet is simply $\{a\}$, and $\widetilde{\mathcal{A}}$ accepts both weakly and strongly monotonic timed words).

4.4 Restricting to a Single State

The final step is to transform $\widetilde{\mathcal{A}}$ into a single-state automaton $\widehat{\mathcal{A}}$ that accepts precisely the same language.

We have already observed that the linearity of $\widetilde{\mathcal{A}}$ places an upper bound on the total number of possible changes between states in any run of the automaton. Equivalently, the transition graph of $\widetilde{\mathcal{A}}$, with all self-loops removed, is simply a directed acyclic graph, or DAG. It follows that one can enumerate the states of $\widetilde{\mathcal{A}}$ as $s_1, s_2, \ldots, s_p$ so that the transition relation is monotone with respect to this enumeration: any transition $s_i \longrightarrow s_j$ of $\widetilde{\mathcal{A}}$ is such that $i \leq j$.[7] Note that this entails that s_p is the sink state.

Let $z_1, z_2, \ldots, z_p$ be p new clocks for $\widehat{\mathcal{A}}$ to use. Our rough intention is to encode state s_k by having, for every $1 \leq i \leq p$, $z_i < 1$ iff $i \leq k$. In order to adequately circumvent various technical difficulties (for instance, in initial states all clocks start with value 0), we refine this correspondence as follows:

1. Initial states are associated with the clock constraint:

$$\phi_{\text{init}} \equiv abs < 1 \vee (abs = 1 \wedge z_1 = 1 \wedge z_2 = 1 \wedge \ldots \wedge z_p = 1).$$

2. The sink state s_p is associated with the clock constraint:

$$\phi_{\text{sink}} \equiv (abs > 1 \wedge ev \geq 1) \vee (abs \geq 1 \wedge z_1 < 1 \wedge z_2 < 1 \wedge \ldots \wedge z_p < 1).$$

3. While $abs < 1$, we do not reset the z-clocks. Afterwards, on any transition, we systematically reset all z-clocks whose values are already strictly less than 1.
4. When entering any inner state s_k, we ensure, for all $1 \leq i \leq p$, that $z_i < 1$ iff $i \leq k$ (which is achieved by resetting all clocks z_i with $i \leq k$ on every transition with target s_k). It is clear that this discipline can be maintained, thanks to the monotonicity of the transition relation with respect to the enumeration of the states, and the fact that all z-clocks, upon leaving an initial state, have value at least 1.
   The clock constraint associated with an inner state s_k is therefore:

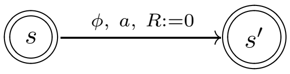
$$\phi_{s_k} \equiv abs \geq 1 \wedge \bigwedge\{z_i < 1 \,|\, i \leq k\} \wedge \bigwedge\{z_i \geq 1 \,|\, i > k\} \wedge \neg\phi_{\text{sink}}.$$

[7] Formally, such an enumeration can be obtained by topologically sorting the underlying transition DAG.

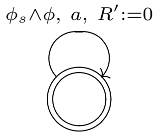
Recall that, after the initialisation phase (i.e., after time 1), consecutive events normally always occur strictly less than 1 time unit apart, otherwise the automaton transitions to the sink state (cf. Property 5 in Subsection 4.2). It therefore follows that, for any inner state s_k, the clock constraint ϕ_{s_k}, which holds upon entering s_k, in fact holds continuously until the next transition occurs, unless that transition is headed for the sink state.

A second important observation is that, once ϕ_{sink} holds, it holds forever. Indeed, the clock ev is never reset in the sink state, so the clock constraint $ev \geq 1$, once true, never changes. On the other hand, if $ev < 1$, then we must have $z_i < 1$ for all i. Moreover, we also have $z_i \leq ev$ for all i, since the z-clocks all had to be reset when the sink state was first entered. It immediately follows that if the z_i's subsequently ever reach 1, then so does ev, so that ϕ_{sink} does indeed hold continuously.

The construction of $\widehat{\mathcal{A}}$ from $\widetilde{\mathcal{A}}$ is now straightforward. All states are merged into one, and every transition



of $\widetilde{\mathcal{A}}$ is replaced by a self-loop



in $\widehat{\mathcal{A}}$, where ϕ_s is the clock constraint associated with state s, and the new reset set R' comprises R together with all clocks z_i that are required to be reset upon entering s'.

It is easy to see that $\widehat{\mathcal{A}}$ and $\widetilde{\mathcal{A}}$ simulate each other's behaviour almost perfectly. The only difference is that $\widehat{\mathcal{A}}$ can 'silently' transition from an initial or an inner state into the sink state, simply through the passage of time (this occurs if clock ev reaches 1). But in such a case, $\widetilde{\mathcal{A}}$'s next transition would take it to the sink state as well, so that there would be no visible difference in terms of behaviour (and language accepted) between the two automata.

We conclude that $\widehat{\mathcal{A}}$ is universal over weakly monotonic time iff the two-counter machine $\mathcal{M}$ does not halt. $\widehat{\mathcal{A}}$ has a single state and a singleton alphabet; moreover, it is straightforward to verify that the only clock constants required in its construction are 0 and 1.

This completes the proof of Theorem 1.

5 Universality over Strongly Monotonic Time

The overall approach to proving the undecidability of universality for single-state, single-event timed automata over strongly monotonic time is very similar to that for the weakly monotonic case. The main difference is that we cannot encode a plurality of events through instantaneous repetitions of a single event; in particular, we have no good way of mimicking the delimiter b. We must therefore alter our encoding of the halting computations of two-counter machines.

Let $\mathcal{M} = (Q, q_0, q_f, C, D, \Xi)$ be a two-counter machine. Recall that in Section 4, configurations of $\mathcal{M}$ were encoded over a unit-duration time interval. The essential difference is that here we encode configurations over three time units, as follows.

Let $s_1, \ldots, s_m$ be an enumeration of $Q \setminus \{q_f\}$, and consider a configuration (s, c, d) of $\mathcal{M}$. Given an integer k, we can encode this configuration over the time interval $[k, k+3)$ using a single event a. More precisely:

1. We encode the state s in the interval $[k, k+1)$, using i occurrences of a if $s = s_i$, and using $m+j$ occurrences of a, for any $j \geq 1$, if $s = q_f$. In addition, we require in both cases that the first a occur at time k.
2. We encode the value c of the first counter in the interval $(k+1, k+2)$ using c occurrences of a.
3. We encode the value d of the second counter in the interval $(k+2, k+3)$ using d occurrences of a.

The construction of a timed language $L'(\mathcal{M})$ containing encodings of the halting computation of $\mathcal{M}$ as strongly monotonic timed words can now proceed along the same lines as before. There is an 'initialisation' phase during the first three time units, followed by an encoding of the computation as successive configurations, and finally an arbitrary suffix. The counter discipline is the same as before, ensuring integrity by copying the contents of the counters exactly three time units apart.

Finally, a single-state, single-event linear safety timed automaton $\widehat{A'}$ can be constructed along the same lines as before to recognise the strongly monotonic complement of $L'(\mathcal{M})$. While not in the sink state, this automaton resets a distinguished clock every three time units; using the constants 1, 2, and 3,[8] it can therefore correctly check timed words for violations, as judged against the encoding described above. We leave the details of the construction to the reader.

This completes the proof sketch of Theorem 2.

6 Concluding Remarks

In this paper, we have studied the undecidability of the universality problem for timed automata with minimal resources, and have obtained some fairly stringent bounds: universality remains undecidable for timed automata with a single state, over a single-event alphabet, and using no more than three distinct clock constants.

One natural question is whether we can further tighten the restriction on clock constants. For example, it is an open problem whether the single constant 1 would suffice, over either weakly or strongly monotonic time.

[8] Note that, over strongly monotonic time, there is no use for the clock constant 0.

References

1. S. Adams. *On the Undecidability of Universality for Timed Automata with Minimal Resources*. MSc Thesis, Oxford University, 2006.
2. R. Alur, C. Courcoubetis, and D. Dill. Model-Checking for Real-Time Systems. In *Proceedings of LICS 90*, pages 414–425. IEEE Computer Society Press, 1990.
3. R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.
4. R. Alur, L. Fix, and T. A. Henzinger. Event-Clock Automata: A Determinizable Class of Timed Automata. *Theoretical Computer Science*, 211:253–273, 1999.
5. R. Alur, S. La Torre, and P. Madhusudan. Perturbed Timed Automata. In *Proceedings of HSCC 05*, volume 3414, pages 70–85. Springer LNCS, 2005.
6. D. Bošnački. Digitization of Timed Automata. In *Proceedings of FMICS 99*, 1999.
7. M. Emmi and R. Majumdar. Decision Problems for the Verification of Real-Time Software. In *Proceedings of HSCC 06*, volume 3927, pages 200–211. Springer LNCS, 2006.
8. A. Finkel and Ph. Schnoebelen. Well-Structured Transition Systems Everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.
9. V. Gupta, T. A. Henzinger, and R. Jagadeesan. Robust Timed Automata. In *Proceedings of HART 97*, volume 1201, pages 331–345. Springer LNCS, 1997.
10. T. A. Henzinger, Z. Manna, and A. Pnueli. What Good Are Digital Clocks? In *Proceedings of ICALP 92*, volume 623, pages 545–558. Springer LNCS, 1992.
11. T. A. Henzinger and J.-F. Raskin. Robust Undecidability of Timed and Hybrid Systems. In *Proceedings of HSCC 00*, volume 1790, pages 145–159. Springer LNCS, 2000.
12. D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. Timed I/O Automata: A mathematical Framework for Modeling and Analyzing Real-Time Systems. In *Proceedings of RTSS 03*. IEEE Computer Society Press, 2003.
13. N. A. Lynch and H. Attiya. Using Mappings to Prove Timing Properties. *Distributed Computing*, 6(2):121–139, 1992.
14. M. L. Minsky. Recursive Unsolvability of Post's Problem of "Tag" and Other Topics in the Theory of Turing Machines. *Annals of Mathematics*, 74(3):437–455, 1961.
15. J. Ouaknine. Digitisation and Full Abstraction for Dense-Time Model Checking. In *Proceedings of TACAS 02*, volume 2280, pages 37–51. Springer LNCS, 2002.
16. J. Ouaknine and J. Worrell. Revisiting Digitization, Robustness, and Decidability for Timed Automata. In *Proceedings of LICS 03*, pages 198–207. IEEE Computer Society Press, 2003.
17. J. Ouaknine and J. Worrell. Timed CSP = Closed Timed ε-Automata. *Nordic Journal of Computing*, 10:99–133, 2003.
18. J. Ouaknine and J. Worrell. Universality and Language Inclusion for Open and Closed Timed Automata. In *Proceedings of HSCC 03*, volume 2623, pages 375–388. Springer LNCS, 2003.
19. J. Ouaknine and J. Worrell. On the Language Inclusion Problem for Timed Automata: Closing a Decidability Gap. In *Proceedings of LICS 04*, pages 54–63. IEEE Computer Society Press, 2004.
20. J.-F. Raskin. *Logics, Automata and Classical Theories for Deciding Real Time*. PhD thesis, University of Namur, 1999.
21. S. Taşiran, R. Alur, R. P. Kurshan, and R. K. Brayton. Verifying Abstractions of Timed Systems. In *Proceedings of CONCUR 96*, volume 1119, pages 546–562. Springer LNCS, 1996.

22. S. Tripakis. Folk Theorems on the Determinization and Minimization of Timed Automata. In *Proceedings of FORMATS 03*, volume 2791, pages 182–188. Springer LNCS, 2003.