On the Language Inclusion Problem for Timed Automata: Closing a Decidability Gap

Joël Ouaknine
Computer Science Department, Carnegie Mellon University
5000 Forbes Ave., Pittsburgh PA 15213, USA
Email: joelo@andrew.cmu.edu

James Worrell
Department of Mathematics, Tulane University
6823 St. Charles Ave., New Orleans LA 70118, USA
Email: jbw@math.tulane.edu

Abstract

We consider the language inclusion problem for timed automata: given two timed automata A and B, are all the timed traces accepted by B also accepted by A? While this problem is known to be undecidable, we show here that it becomes decidable if A is restricted to having at most one clock. This is somewhat surprising, since it is well-known that there exist timed automata with a single clock that cannot be complemented. The crux of our proof consists in reducing the language inclusion problem to a reachability question on an infinite graph; we then construct a suitable well-quasi-order on the nodes of this graph, which ensures the termination of our search algorithm.

We also show that the language inclusion problem is decidable if the only constant appearing among the clock constraints of A is zero. Moreover, these two cases are essentially the only decidable instances of language inclusion, in terms of restricting the various resources of timed automata.

1. Introduction

Timed automata were introduced by Alur and Dill in [5] and have since become a standard modeling formalism for real-time systems. Unfortunately, the algorithmic analysis of timed automata is limited by the undecidability of the language inclusion problem (given two timed automata A and B, are all the timed traces accepted by B also accepted by A?) [5]. In spite of this hindrance, there has been much research in the last decade on various aspects of timed language inclusion—see, e.g., [29], [20], [18], [10], [13], [24], [6], [27], [12], [7], [22], [26], [25]. In this paper, we show that, if the timed automaton A is restricted to having a single clock, the language inclusion question of whether $L(B) \subseteq L(A)$ becomes decidable.

This is somewhat surprising, since the vast majority of decidable instances of language inclusion among both timed and untimed computational models proceed by complementation and emptiness checking of the intersection [16]: $L(B) \subseteq L(A)$ iff $L(B) \cap \overline{L(A)} = \emptyset$. However, it is well-known that there exist timed automata with a single clock that cannot be complemented, which precludes any (direct) use of the above equivalence.

We solve the timed automaton language inclusion problem $L(B) \subseteq L(A)$, in which A is assumed to have at most one clock, by converting it to a *reachability* problem on an infinite 'joint state space' of A and B. This procedure requires us to determinize and complement A *on-the-fly*, creating an unbounded object. Fortunately, we are able to construct a suitable well-quasi-order on the state space, which ensures termination.

We also show that the timed automaton language inclusion problem $L(B) \subseteq L(A)$ is decidable if the only constant appearing among the clock constraints of A is zero (in this case, of course, both timed automata are allowed arbitrarily many clocks). Interestingly, no other set of 'reasonable' restrictions on the *resources* of timed automata (number of clocks, number of locations, magnitude of clock constraints, and size of alphabet) yields a decidable language inclusion problem.

The results presented in this paper paint a fairly complete theoretical picture of the language inclusion problem for timed automata. We believe that they also have promising practical applications, as we now argue.

In software engineering, it is common to have several representations of a system under development, at different levels of abstraction. One of the most widespread abstraction and specification formalisms is that of *finite-state machines*—see, e.g., [11], [19], [21], [8]. The intention is that more concrete representations of the system, including in particular any proposed implementation, should always *conform* to the abstract specification. A standard notion of conformance is that of (untimed) language inclusion: every trace of the system should also be a trace of the specification. Unfortunately, finite-state machines are *time-abstract*, in that they do not incorporate timing details. However, for many systems (such as communication protocols or plant controllers), timing considerations can be crucial to ensure correctness. For this reason, many researchers advocate the use of *timed* finite-state machines to represent specifications, with *timed* language inclusion as the conformance relation between implementation and specification—see, e.g., [29], [10], [6], [25], [18].

Although this notion of conformance between an implementation and a timed specification is easy to state, verifying

This research was sponsored by the Semiconductor Research Corporation (SRC) under contract no. 99-TJ-684, the National Science Foundation (NSF) under grants no. CCR-9803774 and CCR-0121547, the Office of Naval Research (ONR) and the Naval Research Laboratory (NRL) under contract no. N00014-01-1-0796, and the Army Research Office (ARO) under contract no. DAAD19-01-1-0485. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of SRC, NSF, ONR, NRL, ARO, the U.S. Government or any other entity.

whether it holds, as discussed above, is in general undecidable. The main result of this paper, which provides an algorithm to check timed language inclusion between implementations and *single-clock* timed specifications, opens the way to the formal hierarchical modeling and automated verification of a large class of systems; one such example is the protocol TCP, used to transmit information over the Internet, whose functional specification can be given as a finite-state machine equipped with a single clock [17, pages 15–23].

Related work. The first paper to consider the timed automaton language inclusion question $L(B) \subseteq L(A)$ was [5], in which the undecidability of the general case was established. Although the proof was only sketched, it clearly showed that the problem is undecidable even if A is restricted to having two clocks. On the other hand, the paper's *region automaton* construction, drawing on earlier work [4], showed that the problem is decidable if A is not permitted the use of any clock. The remaining case—A having a single clock—has, to the best of our knowledge, never been studied before.

Several researchers have investigated timed automaton language inclusion under various other assumptions. Among others, we note the use of (i) topological restrictions and digitization techniques: [12], [7], [26], [22], [25]; (ii) fuzzy semantics: [10], [13], [24]; (iii) determinizable subclasses of timed automata: [6], [27]; and (iv) timed simulation relations and homomorphisms: [29], [20], [18].

Most decision algorithms for timed automata are based on *clock region* constructions [4], [5]. Clock regions partition the dense (infinite) state space of clocks into *finitely* many pieces, in such a way that the resulting quotient exhibits the same *qualitative* behavior as the original system. Unfortunately, this relationship is not strong enough to preserve *quantitative* properties such as timed language inclusion.

Although the constructions we use in this paper rely in part on clock regions, they give rise in general to objects that are intrinsically *infinite*. We are able to ensure termination of our algorithm by carefully manufacturing and exploiting a suitable *well-quasi-order (wqo)* on our state space. The use of wqos to provide termination guarantees for algorithms that operate on infinite structures is certainly not new: other decidability results include questions of *reachability*, *maintainability*, *termination*, *coverability/sub-coverability of markings* (in Petri nets), and *simulation by/of finite-state machines*. We refer the reader to the excellent surveys [3], [9] for more details on these matters. To our knowledge, however, our work is the first to apply the theory of wqos to a *language inclusion* problem.

The wqo we use in this paper relies on *Higman's lemma* [15] and is obtained through an elaborate process in which, among others, we demonstrate the wqo's compatibility with the decision problem at hand. Other applications of wqos based on Higman's lemma include reachability algorithms for lossy channel systems [1] and parameterized networks of timed processes [2]; additional examples can be found in the two surveys cited earlier.

Structure of the paper. The next section briefly reviews the necessary material on well-quasi-orders and Higman's lemma. Section 3 then carefully presents the model of timed automata we shall use in this paper, along with related definitions and conventions. We also give an example of a single-clock timed automaton that cannot be complemented. In Section 4, we state and prove both of our language inclusion decidability results. Section 5 then presents a number of undecidability results about the universality problem, a special case of language inclusion. Together, Sections 4 and 5 essentially characterize the decidable instances of the language inclusion problem as a function of the resources allocated to timed automata. Lastly, Section 6 offers conclusions and discusses future work.

2. Well-Quasi-Orders and Higman's Lemma

Given a set $\mathcal{Q}$, a *quasi-order*[1] on $\mathcal{Q}$ is a reflexive and transitive relation $\preccurlyeq \subseteq \mathcal{Q} \times \mathcal{Q}$.

An infinite sequence $\langle q_1, q_2, \dots \rangle$ in $\mathcal{Q}$ is said to be *saturating* if there exist indices $i < j$ such that $q_i \preccurlyeq q_j$. A quasi-order $\preccurlyeq$ is a *well-quasi-order* (*wqo* for short) on $\mathcal{Q}$ if every infinite sequence in $\mathcal{Q}$ is saturating.

Let $\sqsubseteq$ be a quasi-order on Λ. Define the induced *monotone domination order* $\preccurlyeq$ on Λ^*, the set of finite words over Λ, as follows: $a_1 \dots a_m \preccurlyeq b_1 \dots b_n$ if there exists a strictly increasing function $f : \{1, \dots, m\} \to \{1, \dots, n\}$ such that, for all $1 \leqslant i \leqslant m$, $a_i \sqsubseteq b_{f(i)}$.

The following result is known as *Higman's lemma* [15]:

Lemma 1: If $\sqsubseteq$ is a wqo on Λ, then the induced monotone domination order $\preccurlyeq$ is also a wqo on Λ^*.

Example 2: Let $\Lambda = \{A, B, \dots, Z\}$ be the standard Roman alphabet, and define the relation $\sqsubseteq$ on Λ to be equality: $x \sqsubseteq y$ iff $x = y$. $\sqsubseteq$ is clearly a wqo since Λ is finite. The induced monotone domination order $\preccurlyeq$ on Λ^* is then none other than the 'subword' order. For example, $HIGMAN \preccurlyeq HIGHMOUNTAIN$ since *HIGMAN* is a subword of *HIGHMOUNTAIN*. Higman's lemma states that $\preccurlyeq$ is a wqo: if one starts writing down an unending sequence of words, one will eventually write down a superword of an earlier word in the sequence.

3. Timed Automata

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

$$\phi ::= \mathbf{true} \mid x \bowtie k \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi.$$

Definition 3: A *timed automaton* is a six-tuple $(\Sigma, S, S_0, S_f, C, E)$, where

- Σ is a finite set (alphabet) of events,
- S is a finite set of locations,

[1] Also sometimes called a *preorder*.

- $S_0 \subseteq S$ is a set of start locations,
- $S_f \subseteq S$ is a set of accepting locations,
- C is a finite set of clocks, and
- $E \subseteq S \times S \times \Sigma \times \Phi_C \times \mathcal{P}(C)$ is a finite set of transitions. A transition (s, s', a, ϕ, R) allows a jump from location s to s', communicating 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.

Remark 4: One finds many variants of the definition of timed automaton in the literature: allowing diagonal clock constraints (of the form $x - y \bowtie k$); allowing rational, rather than integer, bounds in clock constraints; adding invariant clock constraints to locations. It is however not difficult to verify that our main results extend straightforwardly to any combination of these variants.

For the remainder of this section, we are assuming a fixed timed automaton $A = (\Sigma, S, S_0, S_f, C, E)$.

A *clock valuation* is a function $\nu : C \to \mathbb{R}^+$, 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 $x \in C$.

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

A *run* of A is a finite alternating sequence of states and delayed transitions $e = (s_0, \nu_0) \xrightarrow{t_1,\theta_1} (s_1, \nu_1) \xrightarrow{t_2,\theta_2} \dots \xrightarrow{t_n,\theta_n} (s_n, \nu_n)$, where $t_i \in \mathbb{R}^+$ and $\theta_i = (s_{i-1}, s_i, a_i, \phi_i, R_i) \in E$, subject to the conditions:

1) for all $0 \leqslant i \leqslant n-1$, $\nu_i + t_{i+1}$ satisfies ϕ_{i+1}, and
2) for all $0 \leqslant i \leqslant n-1$, $\nu_{i+1}(x) = \nu_i(x) + t_{i+1}$ for all $x \in C \setminus R_{i+1}$, and $\nu_{i+1}(x) = 0$ for all $x \in R_{i+1}$.

Each t_i is interpreted as the time delay between the firing of transitions, and each state (s_i, ν_i), for $i \geqslant 1$, records the data immediately following transition θ_i. We often abuse notation and write runs in the form $(s_0, \nu_0) \xrightarrow{t_1,a_1} (s_1, \nu_1) \xrightarrow{t_2,a_2} \dots \xrightarrow{t_n,a_n} (s_n, \nu_n)$ to highlight the run's events.

An *A-configuration* is a finite set of states of A. Given an A-configuration G, a *G-initialized* run is a run whose first state belongs to G. An *accepting* run, on the other hand, is a run whose last state belongs to S_f.

A *timed event* is a pair (t, a), where $t \in \mathbb{R}^+$ is a delay and $a \in \Sigma$ is an event. A *timed trace* is a finite sequence of timed events, in which each delay represents the time elapsed since the occurrence of the previous event (or since time 0 in the case of the first event). We write $\mathbf{TT}$ to denote the set of all timed traces.

Given a run $e = (s_0, \nu_0) \xrightarrow{t_1,a_1} (s_1, \nu_1) \xrightarrow{t_2,a_2} \dots \xrightarrow{t_n,a_n} (s_n, \nu_n)$, we produce an associated timed trace $\mathtt{tt}(e) \mathrel{\widehat{=}} \langle (t_1, a_1), (t_2, a_2), \dots, (t_n, a_n) \rangle$.

Let G be an A-configuration. We define the *G-initialized timed language* of A to be the set

$$L(A[G]) \mathrel{\widehat{=}} \{\mathtt{tt}(e) \mid e \text{ is an accepting } G\text{-initialized run of } A\}$$

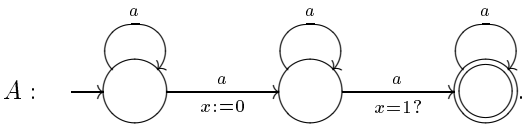
of dense-time timed traces accepted by A, when started in configuration G. A very important special case is that in which $G = S_0 \times \{\mathbf{0}\}$, where $\mathbf{0}$ is the clock valuation mapping every clock to 0. In that case, we write

$$L(A) \mathrel{\widehat{=}} L(A[S_0 \times \{\mathbf{0}\}])$$

to denote the timed language accepted by A (from its standard initial configuration). Another notable instance is that of a singleton A-configuration $G = \{(s, \nu)\}$, in which case we write $L(A[(s, \nu)])$ rather than $L(A[\{(s, \nu)\}])$. Lastly, observe that $L(A[\emptyset]) = \emptyset$.

Remark 5: The reader will have noticed that our timed trace semantics is *weakly monotonic*, in that multiple events are allowed to occur 'simultaneously' (i.e., with no delay between them). None of the results of Section 4 are affected if one adopts instead a *strongly monotonic* semantics, in which all delays are required to be strictly positive. The effects of a strongly monotonic semantics on Theorem 21 in Section 5 are listed in a footnote attached to the statement of the theorem.

Example 6: We reproduce below from [5] an example of a timed automaton[2] A, equipped with a single clock, that cannot be complemented: there does not exist a timed automaton A' such that $L(A') = \mathbf{TT} \setminus L(A)$.



The complement of $L(A)$ contains all timed traces in which no pair of a's is separated by exactly one time unit. Intuitively, since there is no bound on the number of a's that can occur in any unit-duration time interval, any timed automaton capturing the complement of $L(A)$ would require an unbounded number of clocks to keep track of the times of all the a's within the past one time unit. A formal proof that A cannot be complemented is given in [14].

4. Decidable Cases of Language Inclusion

We now present two decidable instances of the language inclusion problem $L(B) \subseteq L(A)$, where A and B are two timed automata. The main result is Theorem 17 in Section 4.1, which asserts that the problem is decidable provided that A is restricted to having at most one clock. Theorem 20 in Section 4.2, on the other hand, states that the problem is also decidable if A does not make use of constants other than 0 in its clock constraints.

[2] Our representation of timed automata follows standard practice: start locations are depicted with an incoming arrow not originating from any other location, and accepting locations are doubly circled. Clock constraints are decorated with question marks (?), whereas clock resets use assignment symbols (:=). The rest of the notation is self-explanatory.

4.1. Single-clock restriction

The main result of this section is Theorem 17, which we present after some preliminaries. We shall assume throughout two fixed timed automata $A = (\Sigma^A, S^A, S_0^A, S_f^A, C^A, E^A)$ and $B = (\Sigma^B, S^B, S_0^B, S_f^B, C^B, E^B)$, with A having a single clock x. Let us moreover postulate, without loss of generality, that A and B share the same alphabet $\Sigma = \Sigma^A = \Sigma^B$, and do not have any other data in common.

The overall strategy for deciding whether $L(B) \subseteq L(A)$ is to explore a certain 'joint state space' of A and B, either making sure throughout that whenever B can accept a particular timed trace then so can A, or otherwise answering the language inclusion query in the negative. As described, this procedure requires that A be determinized, and therefore involves exploring a potentially infinite state space. We ensure termination both by determinizing A *on-the-fly*, as needed, and by constructing a suitable well-quasi-order which forces us only to explore a finite portion of the entire state space.

Since A has only one clock, states of A are simply pairs (s, u), with $s \in S^A$, and $u \in \mathbb{R}^+$ representing the value of clock x. Define an A/B*-configuration* to be a pair $(G, (q, \nu))$, where G is an A-configuration (a finite set of states of A), and (q, ν) is a single state of B.

Intuitively, an A/B-configuration will be used to represent a particular state that B can be in having performed some timed trace π, together with the set of all states that A can be in having performed the same timed trace π. A/B-configurations can therefore be viewed as states of the 'synchronous parallel composition' of A and B, in which A has been determinized.

For (q, ν) a state of B, $t \in \mathbb{R}^+$, and $a \in \Sigma$, let

$$\mathsf{Succ}^B((q,\nu), t, a) \mathrel{\widehat{=}} \{(q', \nu') \mid (q, \nu) \xrightarrow{t,a} (q', \nu')\}$$

be the set of (t, a)-successor states of (q, ν). A similar definition yields a function Succ^A for the timed automaton A, which we lift to A-configurations in the obvious way:

$$\mathsf{Succ}^A(G, t, a) \mathrel{\widehat{=}} \{(s', u') \mid \exists (s, u) \in G \centerdot (s, u) \xrightarrow{t,a} (s', u')\}.$$

Note that $\mathsf{Succ}^A(G, t, a)$ is again an A-configuration, albeit possibly empty.

Let $\Gamma_1 = (G_1, (q_1, \nu_1))$ and $\Gamma_2 = (G_2, (q_2, \nu_2))$ be two A/B-configurations, and let $a \in \Sigma$ be an event. Postulate an *a-transition* from Γ_1 to Γ_2 (written $\Gamma_1 \xrightarrow{a} \Gamma_2$) if there exists $t \in \mathbb{R}^+$ such that $G_2 = \mathsf{Succ}^A(G_1, t, a)$ and $(q_2, \nu_2) \in \mathsf{Succ}^B((q_1, \nu_1), t, a)$; moreover, if $t = 0$ is a valid such witness, we say that the a-transition is *immediate*. In this way, we view the collection of all A/B-configurations as an infinite labeled transition system $\mathcal{G}$. For Γ and Γ' two A/B-configurations, we say that Γ' is *reachable* from Γ if there exists a finite path $\Gamma \xrightarrow{a_1} \ldots \xrightarrow{a_n} \Gamma'$ from Γ to Γ' in $\mathcal{G}$. We include paths of length 0 in this definition, so that any A/B-configuration is reachable from itself.

Let $(G, (q, \nu))$ be an A/B-configuration. We say that $(G, (q, \nu))$ is *bad* if both q is accepting ($q \in S_f^B$), and none of the states in G are accepting (for all $(s, u) \in G$, $s \notin S_f^A$). We also say that $(G, (q, \nu))$ is *doomed* if some bad A/B-configuration is reachable from $(G, (q, \nu))$. In particular, every bad A/B-configuration is doomed. An A/B-configuration is *safe* if it is not doomed.

Lemma 7: For any A/B-configuration $\Gamma = (G, (q, \nu))$, $L(B[(q, \nu)]) \subseteq L(A[G])$ iff Γ is safe.

Proof: Suppose first that Γ is safe, and let $\langle (t_1, a_1), \ldots, (t_n, a_n) \rangle \in L(B[(q, \nu)])$. There is then a corresponding path $\Gamma \xrightarrow{a_1} \Gamma_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} \Gamma_n = (G_n, (q_n, \nu_n))$ in $\mathcal{G}$, where $q_n \in S_f^B$. Since Γ is safe, Γ_n cannot be bad, and therefore there must be some $(s, u) \in G_n$ with $s \in S_f^A$. We conclude that A must have a G-initialized run ending in (s, u) that yields the timed trace $\langle (t_1, a_1), \ldots, (t_n, a_n) \rangle$, which shows that $L(B[(q, \nu)]) \subseteq L(A[G])$ as required.

The other direction proceeds similarly and is left to the reader. ■

Let us call any A/B-configuration of the form $(S_0^A \times \{0\}, (q, \mathbf{0}))$, with $q \in S_0^B$, an *initial* A/B-configuration. (Recall that $\mathbf{0}$ stands for the clock valuation that maps all of B's clocks to 0). We now have:

Corollary 8: $L(B) \subseteq L(A)$ iff all initial A/B-configurations are safe.

Proof: Follows immediately from Lemma 7. ■

Corollary 8 therefore reduces our language inclusion question $L(B) \subseteq L(A)$ to a reachability query on the infinite labeled transition system $\mathcal{G}$. We now construct an equivalence relation on $\mathcal{G}$ by encoding A/B-configurations as words over a certain alphabet. This will enable us to define a suitable well-quasi-order on the resulting quotient labeled transition system.

Let K be the largest constant appearing in any of the clock constraints of A and B. We partition $\mathbb{R}^+$ into a finite collection of one-dimensional regions $REG \mathrel{\widehat{=}} \{r_0, r_1, \ldots, r_{2K+1}\}$, as follows: for $0 \leqslant i \leqslant K$, $r_{2i} \mathrel{\widehat{=}} \{i\}$ and $r_{2i+1} \mathrel{\widehat{=}} (i, i+1)$, and $r_{2K+1} \mathrel{\widehat{=}} (K, \infty)$.

Let $\Lambda \mathrel{\widehat{=}} \mathcal{P}\left((S^A \times REG) \cup (S^B \times C^B \times REG)\right)$ be an alphabet: the 'letters' it contains are finite sets of pairs (s, r) and triples (q, y, r), where s and q are locations of A and B respectively, y is a clock of B, and r is a region. Since Λ, being finite, is clearly well-quasi-ordered by set inclusion, Higman's lemma states that the set Λ^* of finite words over Λ is well-quasi-ordered by the induced monotone domination order $\preccurlyeq$: $\rho_1 \ldots \rho_m \preccurlyeq \gamma_1 \ldots \gamma_n$ if there exists a strictly increasing function $f : \{1, \ldots, m\} \to \{1, \ldots, n\}$ such that, for all $1 \leqslant i \leqslant m$, $\rho_i \subseteq \gamma_{f(i)}$. Note that this order is different from the 'subword' order seen in Example 2.

We now explain how to associate to any A/B-configuration $\Gamma = (G, (q, \nu))$ a canonical word $H(\Gamma) \in \Lambda^*$. Let us assume that the timed automaton B has M clocks $y_1, \ldots, y_M$. If $G = \{(s_1, u_1), \ldots, (s_k, u_k)\}$, we can first equivalently represent Γ as the set

$$\{(s_i, \mathsf{reg}(u_i), \overline{u_i}) \mid 1 \leqslant i \leqslant k\} \cup$$
$$\{(q, y_j, \mathsf{reg}(\nu(y_j)), \overline{\nu(y_j)}) \mid 1 \leqslant j \leqslant M\},$$

where $\mathsf{reg}(t) \in REG$ denotes the region to which the real number $t \in \mathbb{R}^+$ belongs, and $\overline{t} \in [0,1)$ represents the fractional part of t.

Since every pair $(s_i, \mathsf{reg}(u_i))$ and every triple $(q, y_j, \mathsf{reg}(\nu(y_j)))$ corresponds to a (singleton) letter of Λ, we can instead write Γ as

$$\{(\mu_l, v_l) \mid 1 \leqslant l \leqslant k+M\},$$

where each μ_l is one of the Λ-letters in question (of the form $\{(s_i, \mathsf{reg}(u_i))\}$ or $\{(q, y_j, \mathsf{reg}(\nu(y_j)))\}$), and each v_l is its associated fractional part (of the form $\overline{u_i}$ or $\overline{\nu(y_j)}$).

Finally, let us group together Λ-letters whose associated fractional parts are identical, yielding a new set of Λ-letters paired with fractional parts

$$\{(\rho_i, w_i) \mid 1 \leqslant i \leqslant p\}$$

as representation of Γ. Here each ρ_i is a union of μ_l's, and the fractional parts w_i are all distinct; formally: $\rho_i = \bigcup\{\mu_l \mid v_l = w_i\}$, and p is the number of such new pairs, i.e., the total number of distinct fractional parts in Γ. Note that some of the ρ_i's may well still be singletons. We then let

$$H(\Gamma) \mathrel{\widehat{=}} \rho_{i_{z_1}} \rho_{i_{z_2}} \cdots \rho_{i_{z_p}},$$

where $z_1 \ldots z_p$ is the permutation of $1 \ldots p$ that puts $w_{z_1} \ldots w_{z_p}$ in ascending order.

Example 9: Let s_1, s_2 be two locations of the timed automaton A, and let q be a location of the timed automaton B. Suppose that B has two clocks, y_1 and y_2. Let $G = \{(s_1, 0.0), (s_1, 0.3), (s_1, 1.2), (s_2, 0.4), (s_2, 1.0)\}$ be an A-configuration, and let (q, ν) be a state of B, where $\nu(y_1) = 0.8$ and $\nu(y_2) = 1.3$. Finally, let $\Gamma = (G, (q, \nu))$ be an A/B-configuration.

Write r_0 to represent the region $\{0\}$, r_0^1 to represent the region (interval) $(0,1)$, r_1 to represent the region $\{1\}$, and r_1^2 to represent the region (interval) $(1,2)$. Then $H(\Gamma)$ is the 5-letter word

$$\left\{(s_1, r_0), (s_2, r_1)\right\} \left\{(s_1, r_1^2)\right\} \left\{(s_1, r_0^1), (q, y_2, r_1^2)\right\} \left\{(s_2, r_0^1)\right\} \left\{(q, y_1, r_0^1)\right\}.$$

We say that two A/B-configurations Γ and Γ' are *equivalent*, written $\Gamma \sim \Gamma'$, if $H(\Gamma) = H(\Gamma')$. We also say that Γ is *dominated by* Γ', written $\Gamma \preccurlyeq \Gamma'$, if (writing $\Gamma' = (G, (q, \nu))$) there exists $G' \subseteq G$ such that $\Gamma \sim (G', (q, \nu))$. The overloading of $\preccurlyeq$ is justified in view of the following:

Proposition 10: For any A/B-configurations Γ and Γ', $\Gamma \preccurlyeq \Gamma'$ iff $H(\Gamma) \preccurlyeq H(\Gamma')$.

Proof: By straightforward inspection of the relevant definitions. ∎

We earlier showed that the assertion $L(B) \subseteq L(A)$ is equivalent to showing that no bad A/B-configuration is reachable in $\mathcal{G}$. Unfortunately, since there are uncountably many A/B-configurations, it is necessary to reason in terms of Λ-words instead. In the next few propositions, we develop the required machinery to do this.

We begin by showing that $\sim$ is a bisimulation relation:

Proposition 11: For any A/B-configurations Γ_1, Γ_1' and event $a \in \Sigma$, if $\Gamma_1 \sim \Gamma_1'$ then

1) for any Γ_2 such that $\Gamma_1 \xrightarrow{a} \Gamma_2$, there exists Γ_2' with $\Gamma_1' \xrightarrow{a} \Gamma_2'$ and $\Gamma_2 \sim \Gamma_2'$,
2) for any Γ_2' such that $\Gamma_1' \xrightarrow{a} \Gamma_2'$, there exists Γ_2 with $\Gamma_1 \xrightarrow{a} \Gamma_2$ and $\Gamma_2 \sim \Gamma_2'$.

Proof: Let Γ_1, Γ_1' be A/B-configurations such that $\Gamma_1 \sim \Gamma_1'$, and let Γ_2 be an A/B-configuration with $\Gamma_1 \xrightarrow{a} \Gamma_2$. We must show that there exists an A/B-configuration Γ_2' such that $\Gamma_1' \xrightarrow{a} \Gamma_2'$ and $\Gamma_2 \sim \Gamma_2'$.

The transition $\Gamma_1 \xrightarrow{a} \Gamma_2$ can be decomposed into a time evolution from Γ_1 to $\Gamma_1 + t$ (for some $t \in \mathbb{R}$), followed by an immediate transition $\Gamma_1 + t \xrightarrow{a} \Gamma_2$. Here $\Gamma_1 + t$ represents the result of adding t to all clock valuations (of both A and B) in Γ_1.

Write $\Gamma_1 = (G, (q, \nu))$ and $\Gamma_1' = (G', (q', \nu'))$. Since $\Gamma_1 \sim \Gamma_1'$, we have $q = q'$. Moreover, ν and ν' must agree on (i) the integer parts of all clocks (if no greater than K), (ii) whether or not clocks have null fractional part, and (iii) the ordering of the fractional parts of all clocks. It easily follows that there must exist $t' \in \mathbb{R}^+$ such that $\nu + t$ and $\nu' + t'$ are also in similar agreement; moreover, since the relationship $\Gamma_1 \sim \Gamma_1'$ also requires the global matching of the integer and fractional parts of the clock valuations in both G and ν with those in G' and ν', we can in fact find t' such that $\Gamma_1 + t \sim \Gamma_1' + t'$.

The agreement described above between $\nu + t$ and $\nu' + t'$ entails that, for any clock constraint $\phi \in \Phi_{C^B}$, $\nu + t$ satisfies ϕ iff $\nu' + t'$ satisfies ϕ (a formal proof of this fact is an easy structural induction on ϕ). The same of course holds for clock valuations in G and G' with respect to clock constraints in Φ_{C^A}. Consequently, $\Gamma_1 + t$ and $\Gamma_1' + t'$ enable exactly the same transitions of the timed automata A and B.

Let us therefore define Γ_2' to be the A/B-configuration obtained from $\Gamma_1' + t'$ upon immediately taking the same a-transitions as those associated with the jump $\Gamma_1 + t \xrightarrow{a} \Gamma_2$. Observe that, upon taking these transitions, corresponding clocks in $\Gamma_1 + t$ and $\Gamma_1' + t'$ are (in both $\Gamma_1 + t$ and $\Gamma_1' + t'$) either left unchanged, or reset to zero. Since $\Gamma_1 + t \sim \Gamma_1' + t'$, it easily follows that $\Gamma_2 \sim \Gamma_2'$, as required. ∎

Corollary 12: The relation $\sim$ preserves badness, doom, and safety: for any A/B-configurations $\Gamma \sim \Gamma'$, Γ is bad iff Γ' is bad, Γ is doomed iff Γ' is doomed, and Γ is safe iff Γ' is safe.

Proof: The case of badness is immediate, whereas doom and safety follow from the preservation of badness and Proposition 11. ∎

We are therefore only interested in A/B-configurations up to $\sim$-equivalence, and thus define a quotient labeled transition

system $\mathcal{H} \subseteq \Lambda^*$ as follows:

$$\mathcal{H} \mathrel{\widehat{=}} \mathcal{G}/\!\sim \mathrel{\widehat{=}} \{H(\Gamma) \mid \Gamma \text{ is an } A/B\text{-configuration}\},$$

and, for $W_1, W_2 \in \mathcal{H}$ and $a \in \Sigma$, postulate a transition $W_1 \stackrel{a}{\rightarrow} W_2$ if, for all $\Gamma_1 \in H^{-1}(W_1)$ there exists $\Gamma_2 \in H^{-1}(W_2)$ with $\Gamma_1 \stackrel{a}{\rightarrow} \Gamma_2$. Lastly, let

$$\mathcal{H}_0 \mathrel{\widehat{=}} \{H(\Gamma) \mid \Gamma \text{ is an initial } A/B\text{-configuration}\}$$

denote the (finite) set of *initial words* of $\mathcal{H}$.

Corollary 13: For any $W_1, W_2 \in \mathcal{H}$ and $a \in \Sigma$, $W_1 \stackrel{a}{\rightarrow} W_2$ iff there exist A/B-configurations $\Gamma_1 \in H^{-1}(W_1)$ and $\Gamma_2 \in H^{-1}(W_2)$ with $\Gamma_1 \stackrel{a}{\rightarrow} \Gamma_2$.

Proof: Follows immediately from Proposition 11. ■

Given a word $W \in \mathcal{H}$, let

$$\mathsf{Succ}(W) \mathrel{\widehat{=}} \{W' \in \mathcal{H} \mid \exists\, a \in \Sigma \,.\, W \stackrel{a}{\rightarrow} W'\}$$

denote the set of successors of W in $\mathcal{H}$.

Proposition 14: For any word $W \in \mathcal{H}$, the set $\mathsf{Succ}(W)$ is finite and effectively computable.

Proof: Given W, it is easy to construct an A/B-configuration Γ such that $H(\Gamma) = W$. Then, given any $a \in \Sigma$, note that there are only finitely many A/B-configurations Γ' with transition $\Gamma \stackrel{a}{\rightarrow} \Gamma'$ immediately enabled, the list of which can readily be computed.

Next, observe that, for any $t \in \mathbb{R}^+$, $H(\Gamma + t)$ is a word with the same number of letters as W, the finite collection of which is also straightforward to enumerate. For each of these words, and for every event $a \in \Sigma$, computing the immediate a-successors can again be done effectively by simply examining a corresponding A/B-configuration. Note that, according to Corollary 13, the particular choices of A/B-configuration we make to compute successors are unimportant. Since the function H, which converts A/B-configurations back into $\mathcal{H}$-words, in clearly computable, what we have just described is an effective algorithm to generate the set $\mathsf{Succ}(W)$. ■

Next, we show that the wqo $\preccurlyeq$ on $\mathcal{H}$ is a simulation relation:

Lemma 15: Let $W_1, W_1' \in \mathcal{H}$ be two words such that $W_1 \preccurlyeq W_1'$. Then, for any $a \in \Sigma$, $W_2' \in \mathcal{H}$, and transition $W_1' \stackrel{a}{\rightarrow} W_2'$, there exists a word $W_2 \in \mathcal{H}$ such that $W_1 \stackrel{a}{\rightarrow} W_2$ and $W_2 \preccurlyeq W_2'$.

Proof: Let W_1, W_1', and W_2' be as above, and let $\Gamma_1 \in H^{-1}(W_1)$, $\Gamma_1' \in H^{-1}(W_1')$, and $\Gamma_2' \in H^{-1}(W_2')$ be such that there is a transition $\Gamma_1' \stackrel{a}{\rightarrow} \Gamma_2'$. By Corollary 13, it suffices to show there exists $\Gamma_2 \preccurlyeq \Gamma_2'$ such that $\Gamma_1 \stackrel{a}{\rightarrow} \Gamma_2$.

Write $\Gamma_1 = (G_1, (q_1, \nu_1))$, $\Gamma_1' = (G_1', (q_1', \nu_1'))$, and $\Gamma_2' = (G_2', (q_2', \nu_2'))$. Since $\Gamma_1' \stackrel{a}{\rightarrow} \Gamma_2'$, by definition there must be some $t \in \mathbb{R}^+$ such that $G_2' = \mathsf{Succ}^A(G_1', t, a)$ and $(q_2', \nu_2') \in \mathsf{Succ}^B((q_1', \nu_1'), t, a)$. Since $W_1 \preccurlyeq W_1'$, $\Gamma_1 \preccurlyeq \Gamma_1'$, i.e., there exists $G_1'' \subseteq G_1'$ such that $\Gamma_1 \sim (G_1'', (q_1', \nu_1'))$. Write $\Gamma_1'' = (G_1'', (q_1', \nu_1'))$, $G_2'' = \mathsf{Succ}^A(G_1'', t, a)$, and $\Gamma_2'' = (G_2'', (q_2', \nu_2'))$. We then have $\Gamma_1 \sim \Gamma_1''$ and $\Gamma_1'' \stackrel{a}{\rightarrow} \Gamma_2''$. We can therefore invoke Proposition 11 to conclude that there exists an A/B-configuration Γ_2 with $\Gamma_1 \stackrel{a}{\rightarrow} \Gamma_2$ and $\Gamma_2 \sim \Gamma_2''$.

Now notice that, since $G_1'' \subseteq G_1'$, $G_2'' = \mathsf{Succ}^A(G_1'', t, a) \subseteq \mathsf{Succ}^A(G_1', t, a) = G_2'$, and hence $\Gamma_2'' \preccurlyeq \Gamma_2'$. Combining this fact with $\Gamma_2 \sim \Gamma_2''$, we easily see that $\Gamma_2 \preccurlyeq \Gamma_2'$, as required. ■

(Note that $\succcurlyeq$ is also a simulation, but we will not need this.)

Let $W \in \mathcal{H}$ be a word and let $\Gamma \in H^{-1}(W)$ be a corresponding A/B-configuration. We attach the expressions *bad*, *doomed*, and *safe* to W according to whether they respectively apply to Γ. (Note that, in doing so, the particular choice of Γ is unimportant, thanks to Corollary 12.) If W is doomed and if $i \in \mathbb{N}$ is the length of a shortest path from W to a bad word, let us say that W is *i-doomed*. Thus, in particular, bad words are 0-doomed.

Proposition 16: Let $W, W' \in \mathcal{H}$ be two words such that $W \preccurlyeq W'$. If W' is i-doomed, then W is j-doomed for some $j \leqslant i$.

Proof: Follows immediately from Lemma 15 and the following observation: for any A/B-configurations Γ and Γ', if $\Gamma \preccurlyeq \Gamma'$ and Γ' is bad, then so is Γ. ■

Figure 1 gives an algorithm for deciding whether $L(B) \subseteq L(A)$. This algorithm uses two set variables, *ToExplore* and *Explored*, in which to store words. Its correctness is the subject of Theorem 17.

```
let ToExplore = H_0
let Explored = ∅
repeat forever
   repeat
      if ToExplore = ∅ return 'L(B) ⊆ L(A)'
      remove some W from ToExplore
      if W is bad return 'L(B) ⊈ L(A)'
   until ∀V ∈ Explored . V ⋠ W
   let ToExplore = ToExplore ∪ Succ(W)
   let Explored = Explored ∪ {W}.
```

Fig. 1. Algorithm to decide whether $L(B) \subseteq L(A)$

Theorem 17: Let A and B be two timed automata, with A having at most one clock. Then the language inclusion question of whether $L(B) \subseteq L(A)$ is decidable.

Proof: From Corollary 8, we know that $L(B) \subseteq L(A)$ iff all initial words are safe. We now show that the latter is precisely what the algorithm given in Figure 1 decides.

We first observe that the algorithm terminates: indeed, if it did not, since *ToExplore* is always a finite set, an infinite collection $W_1, W_2, \ldots$ of words would over time be added to *Explored*, each new word having the property that it does not dominate any of its predecessors. This would constitute an infinite non-saturating sequence, directly contradicting Higman's lemma.

Next, it is clear that if the algorithm returns '$L(B) \not\subseteq L(A)$', then that statement is accurate: some bad word is reachable from one of the initial words in $\mathcal{H}_0$. On the other hand, if *ToExplore* ever comes to contain a bad word, then the algorithm will inevitably return '$L(B) \not\subseteq L(A)$'.

We now claim that, if *ToExplore* ever comes to contain a doomed word, then eventually the algorithm will also return '$L(B) \not\subseteq L(A)$'. Suppose, on the contrary, that in a given complete execution of the algorithm, the lowest doom index achieved by *ToExplore* is some $i \geqslant 1$; i.e., at some point, an i-doomed word W belonged to *ToExplore*, and for every other word V to have belonged to *ToExplore*, V was either safe or j-doomed, for some $j \geqslant i$. Since W is i-doomed, one of its successors in $\mathsf{Succ}(W)$ must be $(i-1)$-doomed. Thus when W was examined in the inner `repeat` loop, it cannot have satisfied the exit condition $\forall V \in \mathit{Explored} \,.\, V \not\preccurlyeq W$, otherwise $\mathsf{Succ}(W)$ would have been added to *ToExplore*, contradicting our minimal choice of i. It follows that there must have been some word $V \in \mathit{Explored}$ with $V \preccurlyeq W$, from which we deduce, according to Proposition 16, that V is j-doomed for some $j \leqslant i$. But V's presence in *Explored* implies that $\mathsf{Succ}(V)$—which contains a $(j-i)$-doomed word—was at some point added to *ToExplore*. This again contradicts our minimal choice of i and shows that, if any initial word in $\mathcal{H}_0$ fails to be safe, then the algorithm will return '$L(B) \not\subseteq L(A)$', as required. ■

*Remark 18: Why does Theorem 17 fail when A is allowed **two** clocks?* As discussed earlier (and see also Theorem 21), Alur and Dill showed in [5] that the language inclusion problem of whether $L(B) \subseteq L(A)$ is in general undecidable if A is allowed two (or more) clocks. It is therefore instructive to point out where the construction and proof of our single-clock decidability result break down when A is a timed automaton with two clocks.

Recall first that, when A has only one clock, a state of A is a pair (s, u), where s is a location and u is a real number representing the value of A's single clock. When examining a configuration of A—i.e., a finite set of states of A—it is essential to know the ordering of the fractional parts of the clock values of states in the configuration: without this information, it would be impossible to accurately predict how the configuration will tranform as time elapses. In the construction and proof of Theorem 17, we keep track of this ordering by simply reproducing it as the order of the letters in the word that encodes the configuration.

If A is now a timed automaton equipped with *two* clocks x and y, a state of A is a triple (s, u, v), where s is a location and u and v are real numbers representing the values of clocks x and y respectively. A configuration of A is again a finite set of states of A. Note, however, that in order to accurately predict how a given configuration will transform as time elapses, it is necessary at a minimum to know the ordering of the fractional parts of the values of clock x of states in the configuration, as well as the ordering of the fractional parts of the values of clock y of states in the configuration.[3] Notice now that if each state in the configuration is represented by some letter, the ordering of these letters can capture either the ordering of the fractional parts of clock x, or the ordering of the fractional parts of clock y, but not both. It is therefore not possible to encode two-clock configurations as words and at the same time preserve all necessary information to accurately predict how configurations evolve over time.

Naturally, other discrete structures (such as directed graphs with colored edges) could easily be used to encode two-clock configurations and retain the necessary information. But no such structures could then be equipped with a wqo compatible with the *domination* order on configurations, as we now demonstrate.

Let A be a timed automaton with two clocks x and y, and let states and configurations of A be defined as above. Let us say that two A-configurations G and G' are *equivalent* if there is a bijection from G to G' that preserves both the ordering of the fractional parts of clock x of states, and the ordering of the fractional parts of clock y of states.[4] We also say that G is *dominated* by G', written $G \preccurlyeq G'$, if there exists $G'' \subseteq G'$ such that G is equivalent to G''.

It turns out that the domination order for two-clock timed automata is *not* a wqo, so that any hope of guaranteeing termination of an algorithm similar to that presented in Figure 1 is doomed. We illustrate this by exhibiting an infinite non-saturating sequence of two-clock configurations $G_1, G_2, G_3, \ldots$. The control location of all the states in these configurations is the same, and moreover all x and y clock values lie strictly between 0 and 1. As a result, each configuration can be represented as a finite subset of the open unit square $(0,1) \times (0,1)$. For $i \geqslant 1$, configuration G_i consists of $2i+2$ points (states), arranged on i distinct horizontal levels, or lines, in a seesaw manner. Each horizontal level holds two points, and an extra point is added to both the lowest and the highest levels. This sequence of configurations is inspired from an infinite antichain of permutations described in [28].

Rather than give a precise definition of our infinite sequence of configurations, we illustrate in Figures 2 to 4 the configurations G_3, G_4, and G_5, from which the general pattern is easily deduced. Note that dotted lines indicate the various horizontal levels, whereas solid seesaw lines are only used as a visual aid to highlight the general pattern. We leave to the reader the easy task of checking that, for $i \neq j$, $G_i \not\preccurlyeq G_j$, which shows that the sequence $G_1, G_2, G_3, \ldots$ indeed never saturates.

4.2. Null-constant restriction

We now show that the language inclusion question $L(B) \subseteq L(A)$ is decidable even if both A and B are allowed arbitrarily many clocks, provided that A never compares its clocks to any constant other than 0.

A timed automaton is said to be *deterministic* if it has a unique start location, and if, whenever two transitions from a common location are labeled with the same event, then their clock constraints are disjoint.

[3] It is in fact also necessary to know the global ordering of the fractional parts of *all* clock values, but let us disregard this additional burden here.

[4] For simplicity, we are omitting in this definition other requirements such as the preservation of control locations and integral parts of clocks, etc., which have no bearing on our main argument.

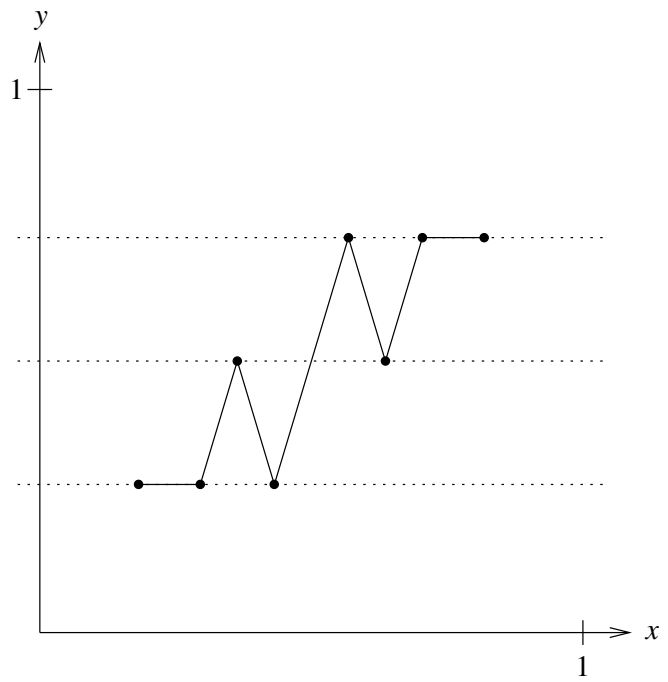


Fig. 2. Configuration G_3 of an infinite non-saturating sequence of configurations of a timed automaton with two clocks

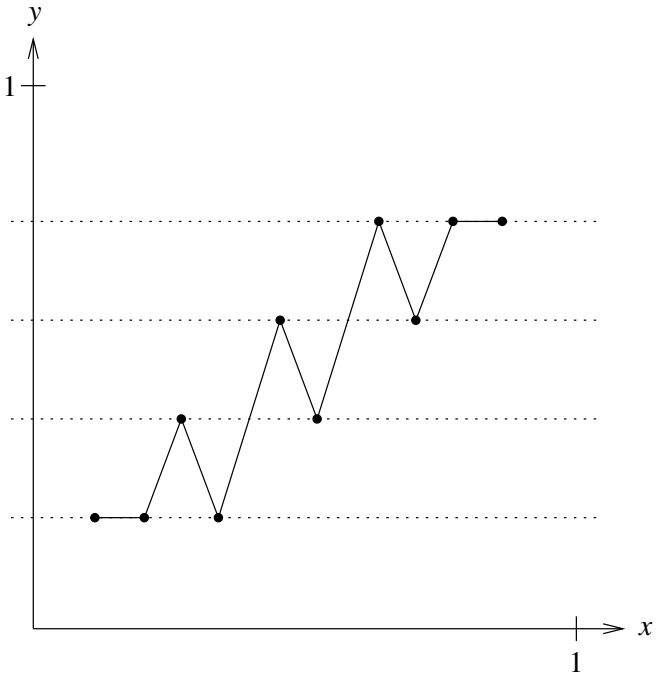


Fig. 3. Configuration G_4

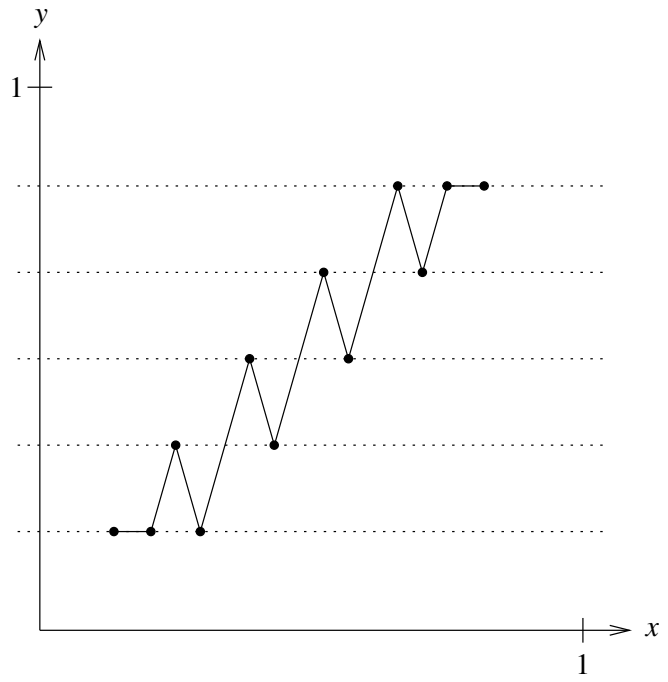


Fig. 4. Configuration G_5

The following result makes use of a construction similar to that given in [30].

Lemma 19: Let A be a timed automaton with 0 the only constant appearing among its clock constraints. Then one can construct a deterministic timed automaton A' which accepts the same timed language: $L(A) = L(A')$. (In addition, A' has a single clock and uses only the constant 0 in its clock constraints.)

Proof: Let A be as above. The idea is to construct a deterministic version of the *region automaton*[5] of A. We will in addition equip this region automaton with a single clock, so as to keep track, on any transition, of whether a strictly positive amount of time has elapsed (since the firing of the last transition) or not. Since A is itself unable to make any finer timed distinctions, the resulting automaton will be equivalent to it.

Let $A = (\Sigma, S, S_0, S_f, C, E)$, with $C = \{x_1, \ldots, x_M\}$ the set of clocks of A. A *clock region* of A is simply an M-tuple of bits, with each bit recording whether its corresponding clock has current value 0 or not. Let REG denote the set of all clock regions. Define a *basic location* to be a pair (s, r), with $s \in S$ a location of A, and $r \in REG$ a clock region. For $a \in \Sigma$, postulate a *basic transition* $(s, r) \stackrel{0,a}{\rightarrow} (s', r')$ if an immediate transition between (s, r) and (s', r') is consistent with some immediate transition of A, and postulate a basic transition $(s, r) \stackrel{1,a}{\rightarrow} (s', r')$ if a delayed transition between (s, r) and (s', r') is consistent with some (strictly positive) time-delayed

[5] The *region automaton* construction, introduced in [5], takes as input a timed automaton A and produces an untimed automaton that accepts the *untimed* language of A: the very same sequences of events, without the delays.

transition of A.

We now construct a deterministic timed automaton A' as follows: its alphabet is the same as that of A, Σ. Its set of locations is $\mathcal{P}(S \times REG)$—in other words, locations of A' are simply sets of basic locations. Its unique start location is $S_0 \times \{\vec{0}\}$, where $\vec{0}$ represents the region consisting entirely of null bits. The accepting locations of A' are those which contain at least one basic location whose first component is accepting (belongs to S_f). A' has a single clock, z, which is reset on every transition. Lastly, for Q, Q' two locations of A' and $a \in \Sigma$, define a transition $Q \xrightarrow{0,a} Q'$ if $Q' = \{(s', r') \mid \exists (s, r) \in Q \centerdot (s, r) \xrightarrow{0,a} (s', r')\}$, and likewise for $Q \xrightarrow{1,a} Q'$. In writing $Q \xrightarrow{1,a} Q'$ we denote the a-labeled transition from Q to Q' which is constrained by $z > 0$ and which subsequently resets z, whereas $Q \xrightarrow{0,a} Q'$ represents the same transition, but constrained by $z = 0$ rather than $z > 0$.

It is readily seen that A' is deterministic, and that it accepts the same timed language as A. The latter rests on the observation that, whenever A accepts a timed trace π, A also accepts any timed trace which is identical to π except for the precise non-zero values of all strictly positive delays. ■

Theorem 20: Let A and B be two timed automata, with 0 the only constant appearing among the clock constraints of A. Then the language inclusion question of whether $L(B) \subseteq L(A)$ is decidable.

Proof: Follows immediately from Lemma 19, the fact that deterministic timed automata can be complemented, the fact that timed automata are closed under intersection, and the well-known fact that language emptiness is decidable [5]. (Alternately, one could directly invoke Theorem 17, since by Lemma 19 A is equivalent to a timed automaton equipped with a single clock.) ■

5. Undecidability of Universality with Minimal Resources

In Section 4, we examined two decidable instances of the language inclusion problem between timed automata. It turns out that these are, for all practical purposes, the *only* decidable instances, at least in terms of placing restrictions on the *resources* of timed automata (number of clocks, number of locations, magnitude of clock constraints, and size of alphabet).

To make this statement more precise, we consider a special case of language inclusion, namely the universality problem (whether a timed automaton accepts every timed trace). For arbitrary timed automata, this problem was shown to be undecidable in [5]. We sharpen this result in the following theorem:

Theorem 21: For A a timed automaton, the universality question of whether $L(A) = \mathbf{TT}$ remains undecidable under any of the following restrictions:

1) A has two clocks and a one-event alphabet[6], **or**
2) A has two clocks and uses a single (non-zero) constant in clock constraints, **or**
3) A has a single location and a one-event alphabet[6], **or**
4) A has a single location and uses a single (non-zero) constant in clock constraints.

[6] Over strongly monotonic time, we require *two* events in A's alphabet.

Remark 22: We recall that diagonal clock constraints (of the form $x - y \bowtie k$) are *not* allowed in our model of timed automata. This restriction considerably complicates cases (3) and (4), since multiple locations cannot simply be encoded through the ordering of clock values, as is otherwise standard [30].

Proof: (Sketch.) In all four cases, the idea of the proof is similar to that presented by Alur and Dill in [5]. Given a two-counter machine M, one constructs a timed automaton A satisfying the relevant restrictions and which moreover rejects precisely those timed traces that correspond (via a certain encoding) to the halting computations of M. It follows that M halts iff $L(A) \neq \mathbf{TT}$. Since the halting problem is undecidable for two-counter machines, so is the universality problem for the corresponding type of timed automata.

Note that Alur and Dill's result imposes no restrictions on timed automata, contrary to Theorem 21. Our encodings and constructions—in particular those pertaining to cases (3) and (4)—are therefore significantly more intricate. Full details can be found in [23]. ■

Note, of course, that the assertion $L(A) = \mathbf{TT}$ reduces to $L(B) \subseteq L(A)$, if B is chosen to be any timed automaton that accepts every timed trace.

An interesting consequence of Theorem 21 (cases (1) and (3)) is that the 'communication' structure of timed automata plays no role in the undecidability of universality. This suggests that the type of questions considered in this paper are no easier to handle in an event-less timed framework than they are here.

6. Conclusion and Future Work

The main contribution of this paper is an algorithm to decide the timed automaton language inclusion question of whether $L(B) \subseteq L(A)$, provided A has at most one clock. We have also shown that the problem is decidable if the only constant appearing among the clock constraints of A is zero. Moreover, these two cases are essentially the *only* decidable instances of language inclusion, in terms of restricting the resources of timed automata.

From a practical point of view, our main decidability result enables the automated verification of timed systems against functional specifications expressed as finite-state machines equipped with a single clock. We believe this to be a substantial improvement in expressiveness over (untimed) finite-state machines, although the feasibility and usefulness of this approach will need to be demonstrated through case studies.

Finally, let us list three interesting directions for future work:

- What is the complexity of our algorithm?
- Can we extend our decidability result to *Büchi* timed automata?
- Are there alternate (e.g., logical) characterizations of the languages accepted by single-clock timed automata?

References

[1] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proceedings of LICS 93*, pages 160–670. IEEE Computer Society Press, 1993.

[2] P. A. Abdulla and B. Jonsson. Verifying networks of timed processes. In *Proceedings of TACAS 98*, volume 1384, pages 298–312. Springer LNCS, 1998.

[3] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proceedings of LICS 96*, pages 313–321. IEEE Computer Society Press, 1996.

[4] 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.

[5] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[6] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999.

[7] D. Bošnački. Digitization of timed automata. In *Proceedings of FMICS 99*, 1999.

[8] S. Chaki, E. M. Clarke, A. Groce, S. Jha, and H. Veith. Modular verification of software components in C. In *Proceedings of ICSE 03*, pages 385–395. IEEE Computer Society, 2003.

[9] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[10] V. Gupta, T. A. Henzinger, and R. Jagadeesan. Robust timed automata. In *Proceedings of HART 97*, volume 1201, pages 331–345. Springer LNCS, 1997.

[11] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[12] 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.

[13] 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.

[14] P. Herrmann. Timed automata and recognizability. *Information Processing Letters*, 65:313–318, 1998.

[15] G. Higman. Ordering by divisibility in abstract algebras. In *Proceedings of the London Mathematical Society*, volume 2, pages 236–366, 1952.

[16] J. E. Hopcroft and J. Ullman. *Introduction to automata theory, languages and computation*. Addison-Wesley, New York, NY, 1979.

[17] Information Sciences Institute, University of Southern California. *Transmission Control Protocol* (DARPA Internet Program Protocol Specification), 1981. `http://www.faqs.org/rfcs/rfc793.html`.

[18] 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.

[19] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines — A survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.

[20] N. A. Lynch and H. Attiya. Using mappings to prove timing properties. *Distributed Computing*, 6(2):121–139, 1992.

[21] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley, 1999.

[22] J. Ouaknine. Digitisation and full abstraction for dense-time model checking. In *Proceedings of TACAS 02*, volume 2280, pages 37–51. Springer LNCS, 2002.

[23] J. Ouaknine and J. B. Worrell. On the undecidability of universality for timed automata with minimal resources. In preparation.

[24] J. Ouaknine and J. B. Worrell. Revisiting digitization, robustness, and decidability for timed automata. In *Proceedings of LICS 03*, pages 198–207. IEEE Computer Society Press, 2003.

[25] J. Ouaknine and J. B. Worrell. Timed CSP = closed timed ε-automata. *Nordic Journal of Computing*, 10:99–133, 2003.

[26] J. Ouaknine and J. B. Worrell. Universality and language inclusion for open and closed timed automata. In *Proceedings of HSCC 03*, volume 2623, pages 375–388. Springer LNCS, 2003.

[27] J.-F. Raskin. *Logics, Automata and Classical Theories for Deciding Real Time*. PhD thesis, University of Namur, 1999.

[28] D. A. Spielman and Miklós Bóna. An infinite antichain of permutations. *The Electronic Journal of Combinatorics*, 7(2):1–4, 2000.

[29] 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.

[30] S. Tripakis. Folk theorems on the determinization and minimization of timed automata. In *Proceedings of FORMATS 03*, 2003.