Fundamenta Informaticae XX (2008) 1–33

IOS Press

Universality Analysis for One-Clock Timed Automata

Parosh Aziz Abdulla
Uppsala University

Johann Deneux
Uppsala University

Joël Ouaknine
Oxford University

Karin Quaas
Leipzig University

James Worrell
Oxford University

Abstract. This paper is concerned with the *universality problem* for timed automata: given a timed automaton $\mathcal{A}$, does $\mathcal{A}$ accept all timed words? Alur and Dill have shown that the universality problem is undecidable if $\mathcal{A}$ has two clocks, but they left open the status of the problem when $\mathcal{A}$ has a single clock. In this paper we close this gap for timed automata over infinite words by showing that the one-clock universality problem is undecidable. For timed automata over finite words we show that the one-clock universality problem is decidable with non-primitive recursive complexity. This reveals a surprising divergence between the theory of timed automata over finite words and over infinite words. We also show that if ε-transitions or non-singular postconditions are allowed, then the one-clock universality problem is undecidable over both finite and infinite words. Furthermore, we present a zone-based algorithm for solving the universality problem for single-clock timed automata. We apply the theory of better quasi-orderings, a refinement of the theory of well quasi-orderings, to prove termination of the algorithm. We have implemented a prototype tool based on our method, and checked universality for a number of timed automata. Comparisons with a region-based prototype tool confirm that zones are a more succinct representation, and hence allow a much more efficient implementation of the universality algorithm.

1. Introduction

Timed automata have emerged as one of the most popular models for specification and analysis of real-time systems. An execution of such an automaton can be viewed as a *timed word* consisting of a sequence of events and their associated timestamps. Furthermore, different properties of the automaton can be expressed as languages of timed words. Since their introduction by Alur and Dill [10], timed automata have been used as the foundation for several verification algorithms and tools (see [14] for a survey). One of the most fundamental results about timed automata is the undecidability of the universality problem: Given a timed automaton $\mathcal{A}$, is the language of $\mathcal{A}$ universal? (In other words, is every timed word accepted by $\mathcal{A}$?). This problem is undecidable when the automaton $\mathcal{A}$ is allowed to have two or more clocks. In this context it is natural to seek subclasses of timed automata, with reduced expressive power, for which universality (or the more general problem of language inclusion) is decidable [12, 15, 14, 11, 23, 33].

A close analysis of the proof of the undecidability of universality (and language inclusion) in [10] reveals that the tightest possible formulation of their result is that the universality problem is undecidable when the automaton has two clocks. This leaves an interesting open question about the status of the one-clock universality problem. In fact, many interesting specifications can be expressed by automata with a single clock, or parallel combinations thereof. This is particularly so for *alternating timed automata* [27, 35]. For instance, every formula of *Metric Temporal Logic* [13, 11] can be translated into an alternating timed automaton with a single clock [35].

Recently, using techniques from the theory of well-quasi-ordered transition systems [1, 22], we showed that over finite words the one-clock universality and language inclusion problems are decidable [34]. However, while finite words are sufficient to capture safety properties, to capture liveness or fairness properties it is most natural to consider automata over infinite words. One result of this paper is that, for timed automata over infinite words (with Büchi acceptance conditions) the one-clock universality problem is undecidable. This reveals a surprising divergence between the theory of timed automata over finite words and over infinite words. We also show that over finite words the one-clock universality problem has non-primitive recursive complexity. The proof here follows the same idea as the proof in [35] that the satisfiability problem for Metric Temporal Logic is non-primitive recursive. Furthermore we show that universality becomes undecidable over both finite and infinite words if ε-transitions are allowed or if clock resets have non-singular postconditions (as in [17, 24]).

We use *channel machines* [19] as a convenient middleware between Turing machines and timed automata. This allows us to develop a schematic approach to proving undecidability and complexity results for various classes of timed automata. In each case we show how to encode a certain class of channel computations as a timed language, whose complement can be recognized by a timed automaton of a certain type.

Despite the positive result in [34] regarding decidability of universality for single-clock timed automata, it is still a difficult task to implement an algorithm for solving the problem. This is illustrated by the hardness result we show in this paper, namely that the problem has non-primitive recursive complexity. In this paper, we consider therefore the challenge of deriving an algorithm which is reasonably efficient on practical examples. In fact, the algorithm in [34] uses a variant of *regions* as a symbolic representation for sets of states in the universality algorithm; and uses the theory of *well quasi-orderings*, for proving termination of the algorithm. It is well-known that the region representation is in general very inefficient and tends to explode even on very small examples. We propose therefore a new for-

malism based on *zones* as a symbolic representation of sets of states in the universality algorithm. Our motivation is twofold. On one hand, several existing verification algorithms for classes of systems with well quasi-ordered state spaces perform well in practice when combined with efficient symbolic representations (despite non-primitive recursive complexities). Examples include lossy channel systems [5], timed Petri nets [8], and parameterized systems [3]. On the other hand, zones often provide a much more compact representation of states than regions. Therefore, zones are used for instance in the design of existing tools for verification of real-time systems, such as KRONOS [38] and UPPAAL [26].

We solve the universality problem by adapting the standard subset construction method. In particular we compute *configurations*: each configuration is the set of states which the automaton reaches through the execution of one timed word. We use zones as symbolic representations of (infinite) sets of configurations. One important aspect of the universality problem is that there is no bound on the number of clock variables in the zones which arise in the analysis. This makes the algorithm much more difficult to design compared to other zone-based algorithms such as the ones used in the above mentioned tools. A main challenge then is to show that the algorithm is still guaranteed to terminate. To achieve this, we show that zones are *well quasi-ordered*. More precisely, we show that, for each infinite sequence of zones $Z_0, Z_1, Z_2, \ldots$, there are i and j with $i < j$ such that the non-universality of Z_j is "entailed" by the non-universality of Z_i. To show the well quasi-ordering of zones, we follow the methodology of [7], and show that zones in fact satisfy a stronger property than well quasi-ordering, namely that they are *better quasi-ordered*.

We have implemented a prototype tool based on our method and have checked a number of timed automata for universality. Comparisons with a region-based prototype tool confirm that zones are a more succinct representation, and hence universality analysis is much more efficient when it operates on zones rather than regions.

We summarize our contributions as follows:

- We show undecidability of the universality problem for one-clock timed automata when the automaton is allowed to have non-singular postconditions or ε-transitions. In the case where we forbid non-singular postconditions and ε-transitions, we show undecidability for the case of infinite words, and non-primitive recursive complexity for the case of finite words.
- We present a zone-based algorithm for solving the universality problem for one-clock timed automata without non-singular postconditions or ε-transitions.

Related Work The non-primitive recursive complexity of language inclusion over finite words and the undecidability of language inclusion over finite words with ε-transitions have recently and independently been proved by Lasota and Walukiewicz [27]. They have also concurrently discovered the undecidability of universality for one-clock Büchi timed automata [28]. Like us, they make use of channel machines in their work, although via a different encoding of channel histories as timed words. (See Remark 4.1 for some explanation of the differences between the respective encodings.)

Alur, La Torre and Madhusudan [15] consider automata with *perturbed clocks* whose rates may vary; they show that for every automaton with a single perturbed clock there is an equivalent deterministic timed automaton. It follows that the language inclusion problem is decidable for this class of automata. Finally, Laroussinie, Markey and Schnoebelen [25] classify the complexity of deciding language emptiness for timed automata with one, two and three clocks respectively.

This article builds upon and extends work that originally appeared in [2] and [9].

Outline In the next section, we give some preliminaries of timed automata. In Section 3, we introduce different classes of channel machines that we use to prove the hardness results. Section 4 shows undecidability in the case where we allow non-singular postconditions or ε-transitions; and also provides an alternative proof of the classical result of undecidability for two-clock timed automata. Section 5 and Section 6 show non-primitive-recursiveness and undecidability in the cases of finite and infinite words respectively. In Section 7 we consider the universality problem for configurations. In Section 8, we introduce zones, and in Section 9, we describe the zone-based universality algorithm. In Section 10, we show that the algorithm is guaranteed to terminate. We devote Section 11 and Section 12 to describe how to implement the different steps of the algorithm; more precisely, we show how to compute successors of zones in the algorithm, and how to check the entailment relation on zones. In Section 13, we report some experimental results. Finally, we give some conclusions and directions for future work in Section 14.

2. Preliminaries

In this section, we recall the basic definitions for timed automata.

We use $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}_+$ to denote the sets of natural numbers, integers, and non-negative reals respectively. For $\delta \in \mathbb{R}_+$, let $\lfloor \delta \rfloor$ and $\mathit{fract}\,(\delta)$ be the integral resp. fractional part of δ.

Timed Words Let Σ be a finite alphabet and write Σ^ε for $\Sigma \cup \{\varepsilon\}$, where $\varepsilon \notin \Sigma$. A *timed event* is a pair (t, a), where $t \in \mathbb{R}_+$ is called the *timestamp* of the *event* $a \in \Sigma$. A *timed word* is a finite or infinite sequence $(t_0, a_0)(t_1, a_1)(t_2, a_2) \cdots$ of timed events whose sequence of timestamps $t_0 t_1 t_2 \ldots t_n$ is non-decreasing and is either finite or diverges to infinity. (This last assumption rules out so-called *Zeno words*.) We say that a timed word is *strictly monotonic* if its sequence of timestamps is strictly increasing. We write $T\Sigma^*$ for the set of finite timed words over alphabet Σ and $T\Sigma^\omega$ for the set of infinite timed words over alphabet Σ.

Timed Automata A timed automaton operates on a finite set C of clocks, denoted c, d, etc. We define the set Φ of *clock constraints* to be conjunctions of formulas of the form $c \sim k$, where $c \in C$, $k \in \mathbb{N}$ and $\sim \in \{<, \leq, >, \geq\}$.

A *timed automaton* is a tuple $\mathcal{A} = (\Sigma, S, s_{init}, F, E)$, where

- Σ is a finite alphabet of events,
- S is a finite set of control states,
- $s_{init} \in S$ is the initial control state,
- $F \subseteq S$ is a set of accepting control states,
- $E \subseteq S \times S \times \Phi \times \Sigma^\varepsilon \times 2^C \times \Phi$ is a finite set of edges. an a-labelled transition from s to s', provided that the precondition $\phi \in \Phi$ on clocks is met. Afterwards, the clocks in R are nondeterministically reset to values satisfying the postcondition ϕ', and all other clocks remain unchanged. Here we assume that only clocks in R are allowed to appear in ϕ'.

We let $cmax$ be the maximum natural number which appears on the edges of the automaton. A *clock valuation* of $\mathcal{A}$ is a function $\nu : C \to \mathbb{R}_+$. A *global state* q of $\mathcal{A}$ is a pair (s, ν), where s is a control state and ν is a clock valuation. We use $state\,(q)$ and $val\,(q)$ to denote s and ν respectively. We say that q is *accepting* if $state\,(q) \in F$. The *initial global state* q_{init} is defined to be (s_{init}, ν_{init}), where $\nu_{init}(c) = 0$ for each $c \in C$.

Suppose that $\mathcal{A} = (\Sigma, S, s_{init}, F, E)$ and $\mathcal{B} = (\Sigma, S', s'_{init}, F', E')$ are timed automata with S and S' disjoint sets. The *union* of $\mathcal{A}$ and $\mathcal{B}$ is obtained by taking the disjoint union of the two automata and adding a new initial state $t \notin S \cup S'$ combining the transitions of s_{init} and s'_{init}. Formally, define $\mathcal{A} \cup \mathcal{B}$ to be the timed automaton $(\Sigma, S \cup S' \cup \{t\}, t, F \cup F', E'')$, where

$$E'' = E \cup E' \cup \{(t, s, \phi, a, R, \phi') : (s_{init}, s, \phi, a, R, \phi') \in E \text{ or } (s'_{init}, s, \phi, a, R, \phi') \in E'\}.$$

Transition Relation We define a transition relation on global states. For a clock valuation ν and $\delta \in \mathbb{R}_+$, we let $\nu + \delta$ be the clock valuation such that $(\nu + \delta)(c) = \nu(c) + \delta$ for all $c \in C$. For a global state q, we let $q + \delta$ be the global state q' such that $state\,(q') = state\,(q)$ and $val\,(q') = val\,(q) + \delta$. A *timed transition* is of the form $q \xrightarrow{\delta}_T q'$, where $q' = q + \delta$. A *discrete transition* is of the form $(s, \nu) \xrightarrow{a}_D (s', \nu')$ such that there is an edge $(s, s', \phi, a, R, \phi')$ in E and the following conditions are met: (i) ν satisfies ϕ; (ii) ν' satisfies ϕ'; and (iii) $\nu'(c) = \nu(c)$ for all $c \in C - R$. We write $q \xrightarrow{\delta, a} q'$ to denote that $q \xrightarrow{\delta}_T q + \delta \xrightarrow{a}_D q'$. For a global state q, a *run* ρ of $\mathcal{A}$ from q is a finite or infinite sequence of transitions

$$\rho = q_0 \xrightarrow{\delta_0, a_0} q_1 \xrightarrow{\delta_1, a_1} q_2 \xrightarrow{\delta_2, a_2} \cdots \tag{1}$$

where $q_0 = q$. We require that an infinite run contain infinitely many transitions labelled from Σ and that $\sum_{i=0}^{\infty} \delta_i$ be infinite. A finite run is *accepting* if the last global state in the run is accepting. An infinite run is accepting if infinitely many global states in the run are accepting. Let $q_0 a_{i_1} a_{i_2} \ldots$ be the sequence of non-ε-labels occurring in an accepting run ρ and let $t_j = \sum_{i=0}^{j} \delta_i$. Then the timed word $(t_{i_0}, a_{i_0})(t_{i_1}, a_{i_1})(t_{i_2}, a_{i_2}) \cdots$ is said to be *accepted along* ρ. Given a global state q we write $L_f(q)$ for the set of finite timed words accepted along finite runs that start in state q. Similarly we write $L_\omega(q)$ for the set of infinite timed words accepted along infinite runs that start in state q. The finite-word timed language of $\mathcal{A}$, denoted $L_f(\mathcal{A})$, is the language $L_f(q_{init})$ accepted from the initial state. Similarly $L_\omega(\mathcal{A})$ denotes the infinite-word timed language $L_\omega(q_{init})$.

The Universality Problem Consider a global state q. In the case of finite words, we say that q is *universal* if $L_f(q) = T\Sigma^*$. Analogously, in the case of infinite words, we say that q is *universal* if $L_\omega(q) = T\Sigma^\omega$. In the *universality problem*, we are given an automaton, and are asked whether the initial global state is universal or not.

Remark 2.1. The above definition represents quite a general model of timed automata. We will adopt the convention that, unless otherwise specified, a given timed automaton has no ε-transitions, and has *singular postconditions*. The last requirement says that clocks that are reset by a transition must be reset to zero: formally for each edge $(s, s', \phi, a, R, \phi')$, ϕ' has the form $\bigwedge_{c \in R} c = 0$.

3. Channel Machines

In this section, we recall the model of a channel machine [4, 19, 36] which consists of a finite-state automaton acting on an unbounded fifo channel (or buffer). We will introduce four attributes for channel machines, namely error-free, lossy, with insertion errors, and balanced. We will later use these to prove hardness results for different classes of timed automata.

Channel Machines A *channel machine* is a tuple $\mathcal{C} = (S, s_{init}, M, \Delta)$, where S is a finite set of *control states*, $s_{init} \in S$ is the *initial control state*, M is a finite set of *messages*, and $\Delta \subseteq S \times L \times S$ is the transition relation over label set $L = \{m!, m? : m \in M\}$. A *global state* of $\mathcal{C}$ is a pair (s, w), where $s \in S$ is the control state and $w \in M^*$ is the contents of the channel. The rules in Δ induce an L-labelled transition relation on the set of global states as follows: $(s, m!, s') \in \Delta$ yields a transition $(s, w) \xrightarrow{m!} (s', w{\cdot}m)$ that writes $m \in M$ to the tail of the channel, and $(s, m?, s') \in \Delta$ yields a transition $(s, m \cdot w) \xrightarrow{m?} (s', w)$ that reads $m \in M$ from the head of the channel. If we only allow the transitions indicated above, then we call $\mathcal{C}$ an *error-free* channel machine.

We also consider channel machines that operate with *insertion errors*. Given $w, v \in M^*$, write $w \sqsubseteq v$ if w can be obtained from v by deleting any number of letters, e.g. $\mathrm{sub} \sqsubseteq \underline{\mathrm{st}}\underline{\mathrm{ub}}\mathrm{born}$, as indicated by the underlining. Following [36] we introduce *insertion errors* by extending the transition relation on global states with the following clause: if $(s, w) \xrightarrow{a} (t, v)$, $w' \sqsubseteq w$ and $v \sqsubseteq v'$, then $(s, w') \xrightarrow{a} (t, v')$. Dually, we define *lossy channel machines* by adding a clause: if $(s, w) \xrightarrow{a} (t, v)$, $w \sqsubseteq w'$ and $v' \sqsubseteq v$, then $(s, w') \xrightarrow{a} (t, v')$.

We say that a channel machine is *balanced* if we can partition the set of control states into two classes, called *read states* and *write states* respectively, such that each edge is of the form $(s, m?, t)$ with s a read state and t a write state, or of the form $(s, m!, t)$ with s a write state and t a read state.

A *computation* of a channel machine $\mathcal{C}$ is a finite or infinite sequence of transitions between global states $(s_0, w_0) \xrightarrow{a_0} (s_1, w_1) \xrightarrow{a_1} (s_2, w_2) \xrightarrow{a_2} \cdots$. Notice that, in particular, if $\mathcal{C}$ is balanced, then any computation consists of an alternating sequence of read transitions and write transitions. An infinite computation of $\mathcal{C}$ is *space-bounded* if there exists $N \in \mathbb{N}$ such that the number of messages stored on the channel during the computation never exceeds N.

(Repeated) Reachability The *control-state reachability problem* asks, given a channel machine $\mathcal{C} = (S, s_{init}, M, \Delta)$ and a control state $t \in S$, whether there is a computation of $\mathcal{C}$ starting in global state (s_{init}, ε) and ending in global state (t, ε). It is well-known that the control-state reachability problem for error-free channel machines is undecidable[1]. The *recurrent-state problem* for channel machines is as follows. Given a channel machine $\mathcal{C} = (S, s_{init}, M, \Delta)$, does there exist $w \in M^*$ such that $\mathcal{C}$ has an infinite computation starting in global state (s_{init}, w) and visiting s_{init} infinitely often? (Henceforth we call such computations *recurrent*.) The *space-bounded recurrent-state problem* is defined in a similar manner, with the difference that we require the infinite computation to be space-bounded.

The following lemma is a relatively straightforward reduction from the halting problem for Turing machines.

[1]The usual formulation of the problem asks whether there is a computation from (s_{init}, ε) to (t, w) for some $w \in M^*$. It is straightforward to reduce this problem to the formulation above.

Lemma 3.1. The recurrent-state problem for error-free balanced channel machines is undecidable.

Given a balanced channel machine $\mathcal{C}$, we claim that $\mathcal{C}$ has a space-bounded recurrent computation with insertion errors iff it has an error-free recurrent computation. Since the channel machine is balanced it is clear that an error-free computation is space-bounded. Conversely an infinite space-bounded computation is eventually error-free since if there were an infinite sequence of insertion errors the computation could not be space-bounded—this again because the machine is balanced. Thus any error-free computation of $\mathcal{C}$ is space-bounded, and any space-bounded computation of $\mathcal{C}$ with insertion errors is eventually error-free (the space bound gives an upper bound on the total number of insertion errors). From this and Lemma 3.1, we get the following lemma, which is reminiscent of a result of Mayr [30] on lossy counter machines.

Lemma 3.2. The space-bounded recurrent-state problem for balanced channel machines with insertion errors is undecidable.

4. Undecidability over Finite Words

In this section, we prove undecidability of the universality problems over finite words for three classes of timed automata, namely those with two clocks, ε-transitions, and non-singular postconditions. The undecidability results are shown through reductions from the reachability problem for (error-free) channel machines. The idea is to first define two timed languages, a language L_{cont} which describes the control part of the channel machine, and another language L_{chan} which captures the channel discipline.

Let $\mathcal{C} = (S, s_{init}, M, \Delta)$ and $t \in S$ be an instance of the control-state reachability problem for channel machines. Given this data, let $\Sigma = \{m!, m? : m \in M\} \cup \{\checkmark\}$ be a finite alphabet. We now define a timed automaton $\mathcal{A}_{cont}$ that intuitively encodes the finite control of $\mathcal{C}$. As with $\mathcal{C}$, $\mathcal{A}_{cont}$ has set of control states S with initial state s_{init}; also $\mathcal{A}_{cont}$ has no clocks. For each transition $(s, m!, t)$ of $\mathcal{C}$ we include in $\mathcal{A}_{cont}$ an $m!$-labelled transition from s to t; similarly for each transition $(s, m?, t)$ of $\mathcal{C}$ we include in $\mathcal{A}_{cont}$ an $m?$-labelled transition from s to t. The only other transition of $\mathcal{A}_{cont}$ are $\checkmark$-labelled self-transitions on every control state. Finally, $t \in S$ is the only accepting control state of $\mathcal{A}_{cont}$. Let L_{cont} denote the timed language $L_f(\mathcal{A}_{cont})$.

Definition 4.1. Let the language $L_{chan} \subseteq T\Sigma^*$ consist of those timed words u such that:

1. u is strictly monotonic.
2. u contains a $\checkmark$-event at time zero, and thereafter consecutive $\checkmark$-events are separated by one time unit.
3. Every $m!$-event in u is followed one time unit later by an $m?$-event.
4. Every $m?$-event is preceded one time unit earlier by an $m!$-event.

Clauses 3 and 4 capture the channel discipline: every message written to the channel is read from the channel one time unit later, and every message that is read from the channel was written to the channel one time unit earlier. The one-to-one unit-time-delayed correspondence between read and write events ensures that messages are read from the channel in the order that they were written to the channel. The

requirement that every message written to the channel is eventually read corresponds to the fact that we consider computations that end with an empty channel. The ✓-events in L_{chan} have no particular signifi cance other than to facilitate the encoding below.

Remark 4.1. Our representation of computations of a channel machine as a timed language is different from that adopted by Lasota and Walukiewicz [28]. They encode a confi guration of the channel machine (control state and channel contents) as a sequence of timed events holding in a unit-length interval along a timed word; successive confi gurations are encoded in successive unit intervals with control states being encoded by propositions with integer timestamps. In contrast, we do not explicitly encode confi gurations, just sequences of reads and writes. Intuitively in our encoding each unit interval corresponds to a *cycle* of the channel, where a cycle is a segment of a computation during which a particular message moves from the tail to the head of the channel. We refer the reader to [18] for a development of this analogy.

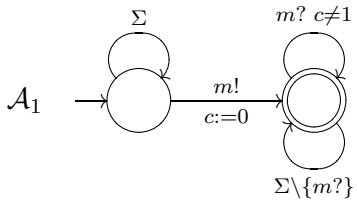
Lemma 4.1. $\mathcal{C}$ has an error-free computation from (s_0, ε) to (t, ε) iff $L_{cont} \cap L_{chan} \neq \emptyset$.

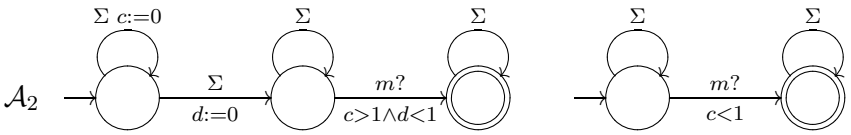
Let $\overline{\mathcal{A}_{cont}}$ denote the complement of $\mathcal{A}_{cont}$ as an untimed automaton. It is clear that $\overline{\mathcal{A}_{cont}}$ is also the complement of $\mathcal{A}_{cont}$ with respect to timed languages, i.e., $L_f(\overline{\mathcal{A}_{cont}}) = T\Sigma^* - L_{cont}$. Now suppose that $\mathcal{A}_{chan}$ is a timed automaton such that $L_f(\mathcal{A}_{chan}) = T\Sigma^* - L_{chan}$. From Lemma 4.1 it holds that $\overline{\mathcal{A}_{cont}} \cup \mathcal{A}_{chan}$ is universal (i.e. accepts every timed word) iff $\mathcal{C}$ has no error-free computation from (s_0, ε) to (t, ε). Since the control-state reachability problem is undecidable for error-free channel machines, it follows that the universality problem is undecidable for any class of timed automata that is closed under unions and can capture the complement of L_{chan}. For each of the three classes of timed automata described below, we show how to defi ne a timed automaton $\mathcal{A}_{chan}$ in the class such that $L_f(\mathcal{A}_{chan}) = T\Sigma^* - L_{chan}$.

4.1. Two clocks

Since $\mathcal{A}_{chan}$ accepts the complement of L_{chan} it is natural to present it as the union of several automata, each of which accepts the set of words that fail to satisfy a particular clause in the defi nition of L_{chan}. We can do this using automata with at most two clocks; the interesting clauses here are 3 and 4.

Automaton $\mathcal{A}_1$, below, accepts those timed words in which some $m!$-event is not followed one time unit later by an $m?$-event, i.e., those words that fail to satisfy Clause 3 in Defi nition 4.1. Automaton $\mathcal{A}_2$ (which is the union of the illustrated left-hand and right-hand components) accepts those timed words in which some $m?$-event is not preceded one time unit earlier by *any* event. Note that a strictly monotonic timed word satisfying Clause 2 in Defi nition 4.1 fails to satisfy Clause 4 if and only if it is either accepted by $\mathcal{A}_2$ or contains an a-event followed one time unit later by an $m?$-event, with $a \neq m!$. It is straightforward to capture this last condition with a one-clock timed automaton. In fact $\mathcal{A}_2$ is the only component of $\mathcal{A}_{chan}$ that uses two clocks.



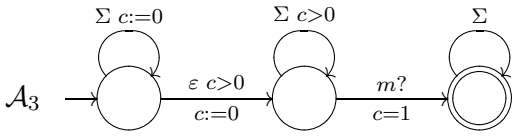


Thus we obtain a new proof of Alur and Dill's classical result [10].

Theorem 4.1. The universality problem for timed automata with two clocks is undecidable.

4.2. ε-transitions

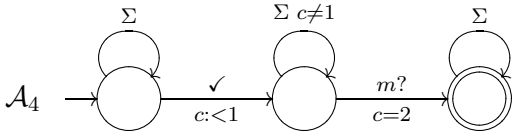
By allowing ε-transitions, we can replace the left-hand component of automaton $\mathcal{A}_2$, above, with the following automaton which uses only one clock. The ε-transition and the $m?$-transition in $\mathcal{A}_3$ are separated by exactly one time unit. Since no visible event happens at the same time as the ε-transition, no visible event precedes this occurrence of $m?$ by exactly one time unit. Thus $\mathcal{A}_3$ accepts precisely those timed words in which there is an event $m?$ with timestamp $t > 1$ such that there is no event with timestamp $t - 1$.



Theorem 4.2. The universality problem for the class of timed automata with one clock and ε-transitions is undecidable.

4.3. Non-singular postconditions

Instead of ε-transitions we can consider non-singular postconditions for clock resets. In this case we can replace automaton $\mathcal{A}_3$ with the following one-clock timed automaton, where the postcondition $c : <1$ on the ✓-labelled edge means that c is non-deterministically reset to a value strictly less than 1.



Theorem 4.3. The universality problem for the class of timed automata with one clock and with non-singular postconditions is undecidable.

5. Non-Primitive Recursiveness over Finite Words

In this section, we show that the universality problem for one-clock timed automata has non-primitive recursive complexity, that is, the problem does not lie in the complexity class $\mathrm{TIME}(f(n))$ for any primitive recursive function $f(n)$. This shows that the problem is very hard since the class of primitive recursive functions includes many fast-growing functions, including non-elementary functions. We show the result through a reduction from the control state reachability problem for lossy channel systems.

The control-state reachability problem for lossy channel machines was shown to be decidable, in contrast to the error-free case, by Abdulla and Jonsson [5]. Later Schnoebelen [36] proved that it has non-primitive recursive complexity. We reduce control-state reachability problem for lossy channel machines to the control-state reachability problem for channel machines with insertion errors. Given a channel machine $\mathcal{C} = (S, s_{init}, M, \Delta)$, define a new transition relation $\Delta^{\mathrm{op}} \subseteq S \times \{m!, m? : m \in M\} \times S$ by replacing each transition $(t, m?, s)$ in Δ by a transition $(s, m!, t)$ in Δ^{op}; and replacing each transition $(t, m!, s)$ in Δ by a transition $(s, m?, t)$ in Δ^{op}. Notice that there is a transition from global state (s, w) to global state (t, v) under Δ iff there is a transition from $(t, rev(v))$ to $(s, rev(w))$ under Δ^{op}, where $rev : M^* \to M^*$ reverses the order of a finite string. Thus there is a computation with lossiness errors from (s, ε) to (t, ε) under Δ iff there is a computation with insertion errors from (t, ε) to (s, ε) under Δ^{op}. This observation allows the desired reduction. From this we get the following.

Lemma 5.1. The control-state reachability problem for channel machines with insertion errors has non-primitive recursive complexity.

Define the timed language $L_{ins} \subseteq T\Sigma^*$ to consist of those timed words satisfying Clauses 1–3 in Definition 4.1. Thus for a word in L_{ins}, every $m!$-event is followed one time unit later by an $m?$-event, but every $m?$-event need not be preceded one time unit earlier by an $m!$-event. This corresponds to a channel with insertion errors.

Lemma 5.2. $\mathcal{C}$ has a computation with insertion errors starting in state (s_0, ε) and ending in state (t, ε) iff $L_{cont} \cap L_{ins} \neq \emptyset$.

Note that we can express $T\Sigma^* - L_{ins}$ as the language of a one-clock timed automaton. This automaton incorporates $\mathcal{A}_1$ in Section 4.1, but not $\mathcal{A}_2$. Thus we obtain

Theorem 5.1. The universality problem for the class of timed automata with a single clock has non-primitive recursive complexity.

6. Undecidability for One-Clock Büchi Automata

In this section we prove the undecidability of the following universality problem: 'Given a one-clock timed automaton $\mathcal{A}$ (without ε-transitions and with singular postconditions) does $L_\omega(\mathcal{A}) = T\Sigma^\omega$?'.

Given a strictly monotonic timed word $u = (t_0, a_0)(t_1, a_1)\ldots$, define $\mathrm{density}(u) = \sup\{j - i : t_j - t_i \leqslant 1\}$. The density of a timed word measures the maximum number of events in any time unit along the word.

As in Sections 4 and 5, the idea behind the proof is to encode the computations of a certain type of channel machine as a timed language. More precisely, the proof is by reduction from the space-bounded recurrent-state problem for balanced machines with insertion errors. We define a timed language $L_{bound} \subseteq T\Sigma^\omega$ encoding space-bounded computations of a channel with insertion errors. We capture the space bound by requiring an upper bound on the number of events per time unit for each word $u \in L_{bound}$. Given a balanced channel machine $\mathcal{C} = (S, s_{init}, M, \Delta)$, we define a one-clock Büchi timed automaton $\mathcal{A}$ such that $\mathcal{C}$ has a space-bounded recurrent computation with insertion errors iff $\mathcal{A}$ is non-universal. Define the finite alphabet $\Sigma = \{m!, m? : m \in M\} \cup \{\checkmark\}$. We encode the finite control of $\mathcal{C}$ as a Büchi timed automaton $\mathcal{A}_{cont}$ with no clocks over alphabet Σ. $\mathcal{A}_{cont}$ is just the underlying

control automaton of $\mathcal{C}$ with a ✓-labelled self-transition added to every control state and with s_{init} as the initial control state and only accepting control state. Let L_{cont} denote the timed language $L_\omega(\mathcal{A}_{cont})$.

Next we capture the behaviour of a space-bounded channel with insertion errors using a timed language L_{bound} over alphabet Σ.

Definition 6.1. L_{bound} consists of those timed words u satisfying:

1. u is strictly monotonic and contains infinitely many non-✓-events.
2. There is a ✓-event at time zero, and thereafter consecutive ✓-events are separated by one time unit.
3. For every $m!$-event in u there is an $m?$-event one time unit later.
4. For every $m?$-event in u there is a $n!$-event one time unit later, for some $n \in M$.
5. $\mathrm{density}(u) < \infty$.

As with the corresponding clause in Definition 4.1, Clause 3 captures the channel discipline: every message sent is received. The channel has insertion errors because not every $m?$-event is necessarily preceded one time unit earlier by an $m!$-event. On the other hand, Clause 4 has nothing to do with the channel discipline. However its presence, together with Clauses 2 and 3, ensures that for every event of $u \in L_{bound}$ there is an event exactly one time unit later. (This fact will play a significant role later.) Since we are dealing with balanced channel machines, the imposition of Clause 4 will prove to be no restriction when we seek to match words in L_{bound} with channel computations. Finally, Clause 5 corresponds to the space-boundedness of the channel.

Lemma 6.1. A balanced channel machine $\mathcal{C}$ has a space-bounded recurrent computation with insertion errors iff $L_{cont} \cap L_{bound} \neq \emptyset$.

Proof:
($\Leftarrow$) Let $u \in L_{cont} \cap L_{bound}$. We show how to recover a space-bounded recurrent computation of $\mathcal{C}$ from u. Since $u \in L_{cont}$, the automaton $\mathcal{A}_{cont}$, which represents the finite control of $\mathcal{C}$, has a run

$$(s_0, \nu_0) \xrightarrow{\delta_0, \alpha_0} (s_1, \nu_1) \xrightarrow{\delta_1, \alpha_1} (s_2, \nu_2) \xrightarrow{\delta_2, \alpha_2} \cdots \tag{2}$$

on u. Let $\alpha_{i_0}\alpha_{i_1}\alpha_{i_2}\ldots$ be the sequence of non-✓-events in u. Then we obtain a recurrent computation of $\mathcal{C}$

$$(s_{i_0}, x_0) \xrightarrow{\alpha_{i_0}} (s_{i_1}, x_1) \xrightarrow{\alpha_{i_1}} (s_{i_2}, x_2) \xrightarrow{\alpha_{i_2}} \cdots$$

where $x_j \in M^*$ is the sequence of messages that occur as read events in the unit time interval $\big(t_{i_{j-1}}, t_{i_{j-1}} + 1\big]$, where, by convention, $i_{-1} = 0$. Since $u \in L_{bound}$, Clause 3 in Definition 6.1 ensures that this is a legitimate computation of $\mathcal{C}$, albeit with insertion errors. Since u has finite density this computation is space-bounded.

($\Rightarrow$) We have already observed that if $\mathcal{C}$ has a space-bounded recurrent computation with insertion errors, then it has a space-bounded recurrent error-free computation. The trace of channel events along such an error-free computation can easily be encoded as a word in L_{bound} as we now explain. Since $\mathcal{C}$ is

balanced, there is a number $N \in \mathbb{N}$ such that the size of the channel is either N or $N-1$ at any point in the computation. When any message is written to the channel, the machine performs exactly $2N-1$ (read and write) operations before that message is read off the channel. We transform the sequence of read and write events along a computation into a timed word u by putting exactly $1/(2N-1)$ time units between consecutive events. This automatically guarantees that Clauses 3–5 in Definition 6.1 hold. Finally, adding ✓-events at integer times yields a timed word in L_{bound}. □

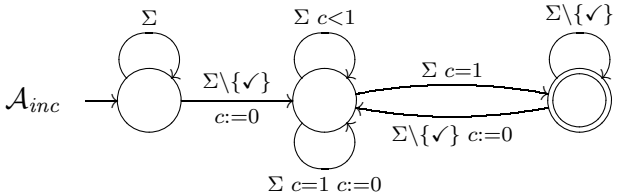
Similarly to the development in Section 4, the undecidability of the universality problem will follow from Lemma 6.1 provided that we can define a one-clock timed automaton $\mathcal{A}_{bound}$ such that $L_\omega(\mathcal{A}_{bound}) = T\Sigma^\omega - L_{bound}$. We define $\mathcal{A}_{bound}$ to be the union of several automata, corresponding to the different clauses in the definition of L_{bound}. It is straightforward, for each clause 1–4, to define an automaton that accepts precisely the timed words that fail to satisfy that clause. Below we define two automata $\mathcal{A}_{inc}$ and $\mathcal{A}_{dec}$ such that, if a timed word u already satisfies 1–4, then it is accepted by $\mathcal{A}_{inc}$ or $\mathcal{A}_{dec}$ precisely if it fails Clause 5, i.e., it has infinite density.

First we recall from [37] the following simple fact about real numbers.

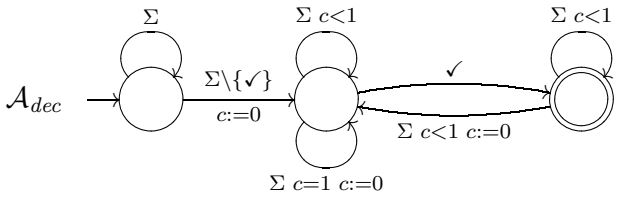
Lemma 6.2. If $\mathbf{x} = \langle x_n : n \in \mathbb{N} \rangle$ is a sequence of real numbers such that $\{x_n : n \in \mathbb{N}\}$ is infinite, then $\mathbf{x}$ has either a strictly increasing subsequence or a strictly decreasing subsequence.

Let $u = (t_0, a_0)(t_1, a_1)(t_2, a_2)\ldots$ be a timed word satisfying Clauses 1–4 in Definition 6.1. Then for every event of u there is an event exactly one time unit later. Thus u has infinite density iff $\{frac(t_i) : i \in \mathbb{N}\}$ is infinite. By Lemma 6.2, this holds iff the sequence $\langle frac(t_i) : i \in \mathbb{N} \rangle$ has either a strictly increasing subsequence or a strictly decreasing subsequence. We define an automaton $\mathcal{A}_{inc}$ that accepts u iff $\langle frac(t_i) : i \in \mathbb{N} \rangle$ has a strictly increasing subsequence, and an automaton $\mathcal{A}_{dec}$ that accepts u iff $\langle frac(t_i) : i \in \mathbb{N} \rangle$ has a strictly decreasing subsequence.

Consider a run of $\mathcal{A}_{inc}$ (depicted below) on $u = (t_0, a_0)(t_1, a_1)(t_2, a_2)\ldots$. Let t_{i_j} be the timestamp of the transition that resets clock x for the j-th time. Notice that either $t_{i_{j+1}} = t_{i_j} + 1$ or $frac(t_{i_{j+1}}) > frac(t_{i_j})$. The Büchi condition ensures that the second eventuality holds infinitely often in the run, and so the sequence $frac(t_{i_j})$ has a strictly increasing subsequence. Thus, among those timed words u satisfying Clauses 1–4 in Definition 6.1, $\mathcal{A}_{inc}$ accepts precisely those for which $\langle frac(t_i) : i \in \mathbb{N} \rangle$ has a strictly increasing subsequence. (Notice the importance of the fact that for each event in u there is an event one time unit later.)



$\mathcal{A}_{dec}$ (depicted below) operates in a similar manner to $\mathcal{A}_{inc}$ except that it accepts those words $u = (t_0, a_0)(t_1, a_1)(t_2, a_2)\ldots$ for which $\langle frac(t_i) : i \in \mathbb{N} \rangle$ has a strictly decreasing subsequence.



We obtain the main result of the section.

Theorem 6.1. The universality problem for one-clock Büchi timed automata is undecidable.

7. Configurations

From this section on, we will concentrate on the universality problem for one-clock timed automata. We fix a timed automaton $(\Sigma, S, s_{init}, F, E)$ that operates on a single clock, henceforth called clock c. For a global state q, we will (abusing notation) use $val\,(q)$ instead of $val\,(q)\,(c)$. In other words, $val\,(q)$ gives the value of (the only) clock c in q.

To solve the universality problem for global states, we study a more general problem, namely the universality problem for (sets of) *configurations*. We will adapt the classical region equivalence to configurations, and show how it can be used to define an entailment relation on configurations.

Configurations A *configuration* γ is a finite set of global states. A configuration γ is said to be *accepting* if some $q \in \gamma$ is accepting. We lift the transition relation from global states to configurations. We use $\gamma \xrightarrow{\delta}_T \gamma'$ to denote that $\gamma' = \left\{q' |\ \exists q \in \gamma.\ q \xrightarrow{\delta}_T q'\right\}$. The definitions of the relations $\xrightarrow{a}_D$ and $\xrightarrow{\delta, a}$ are extended to configurations in a similar manner. For a configuration γ, a *run* (of $\mathcal{A}$) from γ is a finite sequence of transitions

$$\gamma_0 \xrightarrow{\delta_0, a_0} \gamma_1 \xrightarrow{\delta_1, a_1} \gamma_2 \xrightarrow{\delta_2, a_2} \cdots \xrightarrow{\delta_{n-1}, a_{n-1}} \gamma_n \tag{3}$$

where $\gamma_0 = \gamma$. The run is *accepting* if γ_n is accepting. We define $L_f(\gamma)$ in a similar manner to the case of global states. Notice that $L_f(\gamma) = \bigcup_{q\in\gamma} L_f(q)$. In particular, this means that the universality of a global state q is equivalent to the universality of the configuration $\gamma = \{q\}$. We say that γ is *universal* if $L_f(\gamma) = T\Sigma^*$.

A set Γ of configurations is said to be *accepting* if all its members are accepting. We use $\Gamma \xrightarrow{\delta}_T \Gamma'$ to denote that $\Gamma' = \left\{\gamma' |\ \exists \gamma \in \Gamma.\ \gamma \xrightarrow{\delta}_T \gamma'\right\}$. The definitions of the other transition relations are extended analogously. Also the notions of a run, an accepting run, $L_f(\Gamma)$, and universality, are extended in a similar manner to the case of sets of configurations. Notice that $L_f(\Gamma) = \bigcap_{\gamma\in\Gamma} L_f(\gamma)$. In particular, this means that the universality of a configuration γ is equivalent to the universality of the set $\Gamma = \{\gamma\}$. We write $\Gamma \Longrightarrow \Gamma'$ to denote that $\Gamma \xrightarrow{\delta}_T \Gamma'' \xrightarrow{a}_D \Gamma'$ for some δ, a, and Γ''. We define $(\Gamma \Longrightarrow)$ to be the set $\{\Gamma' |\ \Gamma \Longrightarrow \Gamma'\}$.

Region Equivalence For configurations γ and γ', and a bijection $h : \gamma \mapsto \gamma'$, we write $\gamma \equiv_h \gamma'$ to denote that the following conditions are satisfied for each $q, q_1, q_2 \in \gamma$:

- $state\,(q) = state\,(h(q))$.
- $val\,(q) \leq cmax$ iff $val\,(h(q)) \leq cmax$.
- if $val\,(q) \leq cmax$ then $\lfloor val\,(q) \rfloor = \lfloor val\,(h(q)) \rfloor$.
- if $val\,(q) \leq cmax$ then it is the case that $fract\,(val\,(q)) = 0$ iff $fract\,(val\,(h(q))) = 0$.
- if $val\,(q_1) \leq cmax$ and $val\,(q_2) \leq cmax$ then $fract\,(val\,(q_1)) \leq fract\,(val\,(q_2))$ iff $fract\,(val\,(h(q_1))) \leq fract\,(val\,(h(q_2)))$.

We write $\gamma \equiv \gamma'$ to denote that $\gamma \equiv_h \gamma'$ for some h. The relation $\equiv$ is an equivalence, and is a modification of the standard region equivalence on global states. The latter relates (multi-clock) global states, while we here relate sets of global states each with a single clock. The following lemma is an adaptation from the classical theory of timed automata [10].

Lemma 7.1. For configurations γ_1, γ_2, and γ_3, if $\gamma_1 \xrightarrow{\delta,\, a} \gamma_2$ and $\gamma_1 \equiv \gamma_3$, then there is a γ_4 such that $\gamma_3 \xrightarrow{\delta,\, a} \gamma_4$ and $\gamma_2 \equiv \gamma_4$.

Entailment We define an *entailment* relation $\sqsubseteq$ on (sets of) configurations. For configurations γ and γ', we write $\gamma \sqsubseteq \gamma'$ to denote that there is a $\gamma'' \subseteq \gamma'$ such that $\gamma'' \equiv \gamma$. For sets of configurations Γ and Γ', we use $\Gamma \sqsubseteq \Gamma'$ to denote that for each $\gamma' \in \Gamma'$, there is a $\gamma \in \Gamma$ such that $\gamma \sqsubseteq \gamma'$. We write $\Gamma \equiv \Gamma'$ to denote that the following two conditions are satisfied: (i) for each $\gamma' \in \Gamma'$, there is a $\gamma \in \Gamma$ such that $\gamma \equiv \gamma'$; and (ii) for each $\gamma \in \Gamma$, there is a $\gamma' \in \Gamma'$ such that $\gamma' \equiv \gamma$. Notice that $\Gamma \equiv \Gamma'$ implies that both $\Gamma \sqsubseteq \Gamma'$ and $\Gamma' \sqsubseteq \Gamma$. The following lemma follows from Lemma 7.1 and the monotonicity of $\stackrel{\delta,\, a}{\Longrightarrow}$ with respect to inclusion of configurations

Lemma 7.2. Let Γ_1, Γ_2, Γ_3 be sets of configurations. If $\Gamma_1 \Longrightarrow \Gamma_2$ and $\Gamma_3 \sqsubseteq \Gamma_1$, then there is a set of configurations Γ_4 such that $\Gamma_3 \Longrightarrow \Gamma_4$ and $\Gamma_4 \sqsubseteq \Gamma_2$.

For a set Γ of configurations, we define the *rank* $rank(\Gamma)$ of Γ to be the smallest n such there is a sequence $\Gamma_0 \Longrightarrow \Gamma_1 \Longrightarrow \Gamma_2 \Longrightarrow \cdots \Longrightarrow \Gamma_n$, where $\Gamma_0 = \Gamma$, and Γ_n is not accepting. In other words, $rank(\Gamma)$ gives the smallest distance through $\Longrightarrow$ from Γ to a non-accepting set of configurations. If Γ is universal then we define $rank(\Gamma) = \infty$. Notice that $rank(\Gamma) = 0$ iff Γ is not accepting. The following two lemmas relate (non-)universality of a set Γ of configurations to the (non-)universality of its successors.

Lemma 7.3. For a set Γ of configurations, if $0 < rank(\Gamma) < \infty$ then there is a $\Gamma' \in (\Gamma \Longrightarrow)$ such that $rank(\Gamma') < rank(\Gamma)$.

Lemma 7.4. For a set Γ of configurations, Γ is universal iff Γ is accepting and each $\Gamma' \in (\Gamma \Longrightarrow)$ is universal.

Notice that if $\Gamma \sqsubseteq \Gamma'$ and Γ' is not accepting then Γ is not accepting. This, together with Lemma 7.2, implies the following lemma. The lemma shows the relation between the entailment relation and the rank function.

Lemma 7.5. For sets Γ and Γ' of configurations, if $\Gamma \sqsubseteq \Gamma'$ then $rank(\Gamma) \leq rank(\Gamma')$.

8. Zones

We will use zones as a symbolic representation of (infinite) sets of configurations in our universality algorithm for timed automata with single clocks. We assume a one-clock timed automaton $\mathcal{A} = (\Sigma, S, s_{init}, F, E)$. Recall that, although $\mathcal{A}$ operates on a single lock, each configuration may contain several global states. To encode these, we will work with sets of variables which we will use in the definition of zones. For each $s \in S$, we will use a set X^s of variables ranging over $\mathbb{R}_+$. For $x \in X^s$, we use $type\,(x)$ to denote the control state s. Intuitively, a variable $x \in X^s$ is used to model a global state whose control state is s and whose clock value is equal to the value assigned to x. We use X to denote the set $\bigcup_{s \in S} X^s$. We will also assume a special variable x^0. We use x^0 in our zone definitions to model a reference clock whose value is equal to zero. The type of x^0 is of no relevance.

Zones A *zone condition* φ is of the form $y - x \sim k$, where $\sim \in \{\leq, <\}$, $x, y \in X \cup \{x^0\}$, and $k \in \mathbb{Z}$. A *zone* Z is a finite conjunction of zone conditions. We use $Var\,(Z)$ to denote the set of variables in X which occur in Z. Furthermore, we require that, for each $x \in Var\,(Z)$, the zone Z contains a condition of the form $x^0 - x \leq k$ where $k \leq 0$. This condition is to prevent negative clock values in the interpretation of Z. We occasionally consider a zone Z to be a set of zone conditions and write, for instance, $(y - x \sim k) \in Z$ to indicate that $y - x \sim k$ is one of the conjuncts in Z.

We define a total ordering $\lhd$ on elements in the set $\{<, \leq\} \times \mathbb{Z}$ such that $(\sim_1, k_1) \lhd (\sim_2, k_2)$ iff either

- $k_1 < k_2$; or
- $k_1 = k_2$ and either $\sim_1 = <$ or $\sim_2 = \leq$.

We define $(\sim_1, k_1) + (\sim_2, k_2)$ to be $(\sim_3, k_3)$ where $k_3 = k_1 + k_2$ and $\sim_3 = <$ iff either $\sim_1 = <$ or $\sim_2 = <$.

Consider a zone Z, a configuration γ, and a mapping $h : Var\,(Z) \mapsto \gamma$ such that $type\,(x) = state\,(h(x))$ for each $x \in Var\,(Z)$. We extend h such that $val\left(h(x^0)\right) = 0$, i.e., the clock value of the process represented by the special variable x^0 is equal to zero. We write $\gamma \models_h Z$ to denote that

- if $(y - x \sim k) \in Z$ then $val\,(h(y)) - val\,(h(x)) \sim k$.

When there is no risk of confusion, we simplify the notation and write $h(x)$ instead of $val\,(h(x))$. For instance, we write $h(y) - h(x) \sim k$ instead of $val\,(h(y)) - val\,(h(x)) \sim k$.

We write $\gamma \models Z$ to denote that $\gamma \models_h Z$ for some h. We use $[\![Z]\!]$ to denote the set $\{\gamma |\ \gamma \models Z\}$. Intuitively, each variable in $Var\,(Z)$ represents one global state. The configurations in $[\![Z]\!]$ contain global states whose control states are defined by the types of the corresponding variables, and whose clock values are related according to the zone conditions. Since $\lhd$ is total, we can assume without loss of generality that, for each x and y, there is at most one zone condition of the form $y - x \sim k$ in Z (if

$(y - x \sim_1 k_1), (y - x \sim_2 k_2) \in Z$ and $(\sim_1, k_1) \triangleleft (\sim_2, k_2)$, then $y - x \sim_2 k_2$ can be safely removed from Z). In a similar manner to regions (Section 7), our interpretation of zones is different from the standard one (described e.g. in [16]). In the standard interpretation, zones characterize sets of (multi-clock) global states, while in our interpretation a zone characterizes sets of global states each with a single clock.

We say that Z is *accepting* if $[\![Z]\!]$ is accepting, and say that Z is *universal* if $[\![Z]\!]$ is universal. To check that q_{init} is universal, we can check the universality of the zone Z_{init} consisting of the zone conditions $x - x^0 \leq 0$ and $x^0 - x \leq 0$ where $x \in X^{s_{init}}$.

Let $Y = \{x_1, \ldots, x_n\}$ be a set of variables. By $\exists Y.Z$ (or $\exists x_1, \ldots, x_n.Z$) we mean the zone we get from Z by removing all conjuncts which contain a variable in Y. For a set $\mathcal{Z}$ of zones, we use $\exists Y.\mathcal{Z}$ to denote the set $\{\exists Y.Z |\ Z \in \mathcal{Z}\}$.

For zones Z and Z', abusing notation, we use $Z \equiv Z'$ resp. $Z \sqsubseteq Z'$ to denote that $[\![Z]\!] \equiv [\![Z']\!]$ resp. $[\![Z]\!] \sqsubseteq [\![Z']\!]$, and use $rank(Z)$ to denote $rank([\![Z]\!])$. We use $Post(Z)$ to denote a finite set $\mathcal{Z}$ of zones such that $\bigcup_{Z' \in \mathcal{Z}} [\![Z']\!] = ([\![Z]\!] \Longrightarrow)$. In Section 11, we show that such a set exists and is computable.

Stability and Consistency A zone Z is said to be *consistent* if $[\![Z]\!] \neq \emptyset$. Notice that an inconsistent zone is trivially universal.

A zone Z is said to be *stable* if it satisfies the following condition:

- If $(x_2 - x_1 \sim_1 k_1) \in Z$ and $(x_3 - x_2 \sim_2 k_2) \in Z$ then $(x_3 - x_1 \sim_3 k_3) \in Z$ for some $(\sim_3, k_3) \triangleleft (\sim_1, k_1) + (\sim_2, k_2)$.

We can use Floyd's algorithm in the same manner as [21] to compute $Stabilize(Z)$, giving the following lemma.

Lemma 8.1. For each consistent zone Z, we can construct a stable zone, denoted $Stabilize(Z)$, such that (i) $Var\,(Stabilize(Z)) = Var\,(Z)$; and (ii) $\gamma \models_h Stabilize(Z)$ iff $\gamma \models_h Z$ for each γ and h.

Notice that $[\![Stabilize(Z)]\!] = [\![Z]\!]$. For a set $\mathcal{Z}$ of zones, we define $Stabilize(\mathcal{Z}) = \{Stabilize(Z) |\ Z \in \mathcal{Z}$ and Z is consisten
For a stable zone Z, we can check whether Z is consistent by checking whether there is a condition of the form $(x - x \sim k) \in Z$ where $(\sim, k) \triangleleft (<, 0)$.

9. Algorithm

The zone-based universality algorithm is defined as follows:

Algorithm 1: Zone-Based Universality Checking
Input: A zone Z_{init}.
Output: Is Z_{init} universal?

```
ToExplore := {Z_init}
Explored := ∅
while ToExplore ≠ ∅
    remove some Z from ToExplore
    if Z is not accepting then
        return (false)
    else if ∃Z' ∈ Explored. Z' ⊑ Z then
        discard Z
    else
        ToExplore := ToExplore ⋃ Post(Z)
        Explored := {Z} ⋃ {Z' | Z' ∈ Explored ∧ (Z ⋢ Z')}
return (true)
end
```

The algorithm inputs a zone Z_{init}, and should check whether Z_{init} is universal or not. The algorithm maintains two sets of zones: a set `ToExplore`, initialized to $\{Z_{init}\}$, of zones that have not yet been analyzed; and a set `Explored`, initialized to the empty set, of zones that contains information about the set of zones that already have been analyzed. The algorithm preserves the following two invariants:

- some zone in (`ToExplore` $\bigcup$ `Explored`) is non-universal iff Z_{init} is non-universal; and
- If Z_{init} is non-universal, then $\exists Z \in$ `ToExplore`. $\forall$Z$' \in$ `Explored`. $rank($Z$) < rank($Z$')$.

Due to the invariants, the following two conditions can be checked during each step of the algorithm:

- if `ToExplore` becomes empty then the algorithm terminates with a positive answer; and
- if a non-accepting zone is detected then the algorithm terminates with a negative answer.

If neither of the two conditions is satisfied, the algorithm proceeds by picking and removing a zone Z from `ToExplore`. Two possibilities arise depending on the value of Z:

- If there exists a zone $Z' \in$ `Explored` with $Z' \sqsubseteq Z$, then we discard Z. The first invariant is preserved by Lemma 7.5. If Z_{init} is non-universal, then the second invariant and Lemma 7.5 imply that there is still some $Z'' \in$ `ToExplore` such that $rank(Z'') < rank(Z') \leq rank(Z)$. This means that the second invariant will also be preserved by this step.
- Otherwise, we generate the zones in $Post(Z)$ and put them in `ToExplore`. The first invariant will be preserved by Lemma 7.4, while the second invariant will be preserved by Lemma 7.3 and Lemma 7.5. Furthermore, we remove all zones in `Explored` which are larger than Z with respect to $\sqsubseteq$. This operation preserves the second invariant trivially while it preserves the second invariant by Lemma 7.5.

Partial correctness of the algorithm follows immediately from the invariant. It remains to show that:

- The algorithm terminates (done in Section 10).
- We can compute $Post$ and can check the entailment relation $\sqsubseteq$ on zones (done in Section 11 and Section 12).

10. Termination

Using the methodology of [1] it can be shown that the universality algorithm of Section 9 is guaranteed to terminate in case $\sqsubseteq$ is a *well quasi-ordering (WQO)*. Following the framework of [7], we show that $\sqsubseteq$ in fact satisfies a stronger property than WQO; namely that it is a *better quasi-ordering (BQO)*.

WQOs A *quasi-ordering*, or a *QO* for short, is a pair $(A, \preceq)$ where $\preceq$ is a reflexive and transitive (binary) relation on a set A. A QO $(A, \preceq)$ is a *well quasi-ordering*, or a *WQO* for short, if for each infinite sequence $a_1, a_2, a_3, \ldots$ of elements of A, there are $i < j$ such that $a_i \preceq a_j$. For a set $B \subseteq A$, we define $\min(B)$ to be a subset of B which satisfies the following two properties:

- for each $a \in B$ there is a $b \in \min(B)$ with $b \preceq a$.
- the elements of $\min(B)$ are not related by $\preceq$, i.e., there are no $a, b \in \min(B)$ with $a \preceq b$.

If there are several sets satisfying the above two conditions, then we assume that $\min(B)$ gives an arbitrary (but fixed) such set. Notice that if $\preceq$ is a WQO then $\min(B)$ is finite.

BQOs In this paragraph, we introduce the basic definitions of better quasi-orderings (BQOs). The definition of BQOs is quite technical and is not strictly needed for understanding the rest of the paper. Nevertheless, we include the definition for the sake of completeness.

We let $\mathbb{N}^{<\omega}$ ($\mathbb{N}^{\omega}$) denote the set of finite (infinite) strictly increasing sequences over the set $\mathbb{N}$ of natural numbers. For $s \in \mathbb{N}^{<\omega}$, we let $\lambda(s)$ be the set of natural numbers occurring in s, and if s is not empty then we let $tail(s)$ be the result of deleting the first element of s. For $s_1 \in \mathbb{N}^{<\omega}$ and $s_2 \in \mathbb{N}^{<\omega} \cup \mathbb{N}^{\omega}$, we write $s_1 \ll s_2$ to denote that s_1 is a proper prefix of s_2. If s_1 is not empty then we write $s_1 \ll_* s_2$ to denote that $tail(s_1) \ll s_2$. An infinite set $\beta \subseteq \mathbb{N}^{<\omega}$ is said to be a barrier if the following two conditions are satisfied.

- There are no $s_1, s_2 \in \beta$ such that $\lambda(s_1) \subsetneq \lambda(s_2)$.
- For each $s_2 \in \mathbb{N}^{\omega}$ there is $s_1 \in \beta$ with $s_1 \ll s_2$.

Let $(A, \preceq)$ be a quasi-ordering. An *A-pattern* is a mapping $f : \beta \mapsto A$, where β is a barrier. We say that f is *good* if there are $s_1, s_2 \in \beta$ such that $s_1 \ll_* s_2$ and $f(s_1) \preceq f(s_2)$. We say that $(A, \preceq)$ is a *better quasi-ordering (bqo)* if each A-pattern is good.

Properties We give the properties of WQOs and BQOs that we use in our termination proof. Given a QO $(A, \preceq)$, we define a QO $(A^*, \preceq^*)$ on the set A^* such that $x_1\ x_2\ \cdots\ x_m \preceq^* y_1\ y_2\ \cdots\ y_n$ if and only if there is a strictly monotonic[2] injection h from $\{1, \ldots, m\}$ to $\{1, \ldots, n\}$ such that $x_i \preceq y_{h(i)}$ for each

[2] Strict monotonicity means that $i < j$ implies $h(i) < h(j)$.

$i : 1 \leq i \leq m$. We define the relation $\preceq^{\mathcal{P}}$ on the set $\mathcal{P}(A)$ of finite subsets of A, so that $A_1 \preceq^{\mathcal{P}} A_2$ if and only if $\forall b \in A_2 : \exists a \in A_1 : a \preceq b$.

Lemma 10.1. For sets $A_1, A_2 \subseteq A$, we have $A_1 \preceq^{\mathcal{P}} A_2$ iff $\min(A_1) \preceq^{\mathcal{P}} \min(A_2)$.

In the following lemma we state some properties of WQOs and BQOs [7, 29].

Lemma 10.2.

1. Each BQO is WQO.
2. If A is finite then $(A, =)$ is BQO.
3. If $(A, \preceq)$ is BQO then $(A^*, \preceq^*)$ is BQO.
4. If $(A, \preceq)$ is BQO then $\left(\mathcal{P}(A), \preceq^{\mathcal{P}}\right)$ is BQO.
5. If $(A, \preceq_1)$ is BQO and $\preceq_1 \subseteq \preceq_2$ then $(A, \preceq_2)$ is BQO.

(Sets of) Configurations are BQO. Fix an automaton $\mathcal{A} = (\Sigma, S, s_{init}, F, E)$. For a global state q, we define the *signature* $sign\,(q)$ of q to be a pair $(s, k) \in S \times \{0, 1, 2, \ldots, 2 \cdot cmax + 1\}$, where $s = state\,(q)$ and k is defined as follows:

- $k = 2 \cdot \lfloor val\,(q) \rfloor$ if $val\,(q) \leq cmax$ and $fract\,(val\,(q)) = 0$.
- $k = 2 \cdot \lfloor val\,(q) \rfloor + 1$ if $val\,(q) < cmax$ and $fract\,(val\,(q)) > 0$.
- $k = 2 \cdot cmax + 1$ if $val\,(q) > cmax$.

For a configuration γ, we define $sign\,(\gamma)$ to be the (unique) word over $\mathcal{P}(S \times \{0, 1, \ldots, 2 \cdot cmax + 1\})$ of the form $r_0 r_1 \cdots r_n$ such that the following properties are satisfied:

- $\{sign\,(q) \mid q \in \gamma\} = r_0 \cup r_1 \cup \cdots \cup r_n$.
- If $q \in r_i$ and $q' \in r_j$ then $fract\,(q) \leq fract\,(q')$ iff $i \leq j$.

The signature can be viewed as an encoding of the region to which the configuration belongs. The ordering among the sets inside the word reflects the relative ordering of the fractional parts. The control states, the integral parts of the clock values, and whether the fractional part is equal to zero, are all stored inside the signature of each global state. Observe that a signature is not an exact encoding of region, as the former keeps track of the fractional parts of clocks greater than $cmax$, while a region equates all such clock values. We define an ordering on configurations induced by signatures as follows. Consider configurations γ and γ' such that $sign\,(\gamma) = r_0 r_1 \cdots r_m$ and $sign\,(\gamma') = r'_0 r'_1 \cdots r'_n$. We use $\gamma \preceq \gamma'$ to denote that there is a strictly monotonic injection $h : \{0, \ldots, m\} \mapsto \{0, \ldots, n\}$ such that $r_i \subseteq r'_{h(i)}$ for each $i : 0 \leq i \leq m$. The above mentioned relation between regions and signatures is captured in the following lemma (a formal proof can be given in a similar manner to [6] or [34]).

Lemma 10.3. For configurations γ and γ' if $\gamma \preceq \gamma'$ then $\gamma \sqsubseteq \gamma'$.

We observe that the signature of each configuration is a finite word over finite sets over a finite alphabet (namely finite sets over $S \times \{0, 1, 2, \ldots, 2 \cdot cmax + 1\}$). Consequently, Lemma 10.2 (Property 2 and Property 3) gives the following:

Lemma 10.4. $\preceq$ is a BQO on the set of configurations.

From Lemma 10.3, Lemma 10.4, and Lemma 10.2 (Property 5) we get the following:

Corollary 10.1. $\sqsubseteq$ is a BQO on the set of configurations.

From the definition of $\sqsubseteq$ on zones, Corollary 10.1, Lemma 10.1, and Lemma 10.2 (Property 4) we get the following

Lemma 10.5. $\sqsubseteq$ is a BQO on zones.

Lemma 10.5 and Lemma 10.2 (Property 1) give the following:

Corollary 10.2. $\sqsubseteq$ is a WQO on zones.

11. Computing Successors

In this section, we show how to compute $Post(Z)$ for some zone Z. We compute $Post(Z)$ as $Post_D(Post_T(Z))$, where $Post_T$ and $Post_D$ characterize timed resp. discrete successors of Z.

Timed Successors For a zone Z, we let $Post_T(Z)$ denote the zone Z' such that $[\![Z]\!] \xrightarrow{\delta}_T [\![Z']\!]$ for some $\delta \in \mathbb{R}_+$. In other words, Z' characterizes the set of configurations which are timed successors of configurations in $[\![Z]\!]$. To compute Z', we first compute the zone Z'' where Z'' is stable and where $[\![Z'']\!] = [\![Z]\!]$ (Lemma 8.1). We can derive $Post_T(Z)$ from Z'' by deleting all clock conditions of the forms $x - x^0 \sim k$. This gives the following:

Lemma 11.1. For a zone Z, we can compute $Post_T(Z)$.

Discrete Successors Fix a timed automaton $\mathcal{A} = (\Sigma, S, s_{init}, F, E)$ and a zone Z. Informally, the idea of computing $Post_D(Z)$ is as follows. We recall that each variable $x \in Var(Z)$ represents one global state q in a configuration $\gamma \in [\![Z]\!]$. The global state q (represented by x) produces a (possibly empty) set of successors. More precisely, each edge $e = (s, s', \phi, a, R, \phi')$ which "matches" x may produce a successor global state q'. Here, x and e are considered to be matching if $type(x)$ is identical to the source control state s in e. Notice that a successor is generated only if $val(q)$ satisfies ϕ. In this manner, a configuration γ produces a set of successors, reflecting the different successors of the individual global states in γ. We formalize the above reasoning in a number of steps.

First, we define the set of matching variables and edges. For a variable $x \in Var(Z)$ and a label $a \in \Sigma$, we let $E(x, a)$ be the set of edges whose source control state is $type(x)$ and whose label is a. For an $a \in \Sigma$, we define the set $Z \odot a = \{(e, x) \mid x \in Var(Z) \wedge e \in E(x, a)\}$. For each pair $(e, x) \in (Z \odot a)$, we use a fresh variable $y_{(e,x)}$ (i.e., $y_{(e,x)}$ is not a member of $Var(Z)$). We define $type\left(y_{(e,x)}\right)$ to be the target control state of e. Intuitively, for $e = (s, s', \phi, a, R, \phi')$, the set $Z \odot a$

contains all pairs (e, x) which are matching, i.e., $type\,(x) = s$. Each such pair can potentially generate a new global state, represented by a new variable $y_{(e,x)}$ in $Post_D(Z)$. Since the control state of the new global state will be s', the type of $y_{(e,x)}$ is also defined to be s'.

For $(e, x) \in Z \odot a$ with $e = (s, s', \phi, a, R)$, we define $Z \otimes (e, x)$ to be one of the following sets:

- if $c \notin R$ then $Z \otimes (e, x) = \{(y_{(e,x)} = x) \wedge \phi(x)\ \ ,\ \neg\phi(x)\}$.
- if $c \in R$ then $Z \otimes (e, x) = \{(y_{(e,x)} = x^0) \wedge \phi(x)\ \ ,\ \neg\phi(x)\}$.

Intuitively, for each pair (e, x), there are two possibilities: either (i) the guard ϕ is satisfied, in which case we generate a new global state represented by the new variable $y_{(e,x)}$ in $Post(Z)$; or (ii) ϕ is not satisfied in which case no new variable is added to $Post(Z)$. If a new global state is added then, depending on whether $c \in R$, there are two possibilities: either (i) if $c \notin R$ then its clock value is equal to the clock value of the original global state; and (ii) if $c \in R$ then its clock value is equal to 0. In the first case we add the condition $y_{(e,x)} = x$, while in the second case we add the condition $y_{(e,x)} = x^0$.

For $a \in \Sigma$, we define $Z \otimes a$ to be the set of zones of the form

$$\left(\bigwedge_{(e,x) \in (Z \odot a)} \phi_{(e,x)} \right) \wedge Z$$

where $\phi_{(e,x)} \in (Z \otimes (e, x))$ for each $(e, x) \in (Z \odot a)$. Finally, we define:

$$Z \oplus a = \ \ \exists Var\,(Z)\,.Stabilize\,(Z \otimes a)$$

Each member of $Z \otimes a$ is a zone which represents the conjunction of the original zone Z with one of the zones in $Post_D(Z)$. To obtain the new zone, we abstract from the variables of Z. The purpose of stabilization is to avoid losing information when removing the elements of $Var\,(Z)$. The following lemma shows correctness of the above construction.

Lemma 11.2. $Post_D(Z) = \bigcup_{a \in \Sigma} Z \oplus a$.

12. Checking Entailment

In this section, we describe how to implement the entailment relation $\sqsubseteq$ on zones.

To characterize entailment, we use formulas in a decidable logic, called *Difference Bound Logic (DBL)*. The atomic formulas are of the form $y - x \sim k$, where x and y are variables interpreted over $\mathbb{R}_+$ and $k \in \mathbb{N}$. Furthermore the set of formulas is closed under the propositional connectives. Satisfiability of DBL-formulas is NP-complete [32]. NP-hardness follows by reducing the satisfiability problem for Boolean formulas. We represent each atomic proposition p by two variables x_p and y_p in the DBL formula. We replace each occurrence of p in the Boolean formula by the atomic formula $x_p - y_p \leq 0$. For NP-easiness, a non-deterministic algorithm works by guessing which zone conditions are true and which are false. A linear time test can check that the guess makes the entire formula true. A polynomial time test can check that the corresponding set of constraints on the reals is in fact satisfiable. The satisability of a conjunction of atomic formulas (a special case of linear programming) can be solved in cubic time using the Floyd-Warshall algorithm [20].

To give the characterization of entailment, we define the notion of areas. An *area condition* ψ is either of the form $(y - x \sim k) \vee (x > cmax - k)$ where $k \geq 0$; or of the form $(y - x \sim k) \vee (x > cmax)$ where $k < 0$ Given a zone condition $\varphi = (y - x \sim k)$, we use $\varphi^{\odot}$ to denote the area condition $(y - x \sim k) \vee (x > cmax - k)$ if $k \geq 0$, and the area condition $(y - x \sim k) \vee (x > cmax)$ if $k < 0$. An *area* A is a conjunction of area conditions. For a zone Z, we use $Z^{\odot}$ to be the area $\bigwedge_{\varphi \in Z} \varphi^{\odot}$. Given a zone Z with $Var(Z) = \{x_1, \ldots, x_m\}$, it is sometimes convenient to view Z as a predicate $Z(x_1, \ldots, x_m)$ on the set $\mathbb{N}^m$ (replacing any occurrence of x^0 by 0). Observe that $\gamma \models_h Z$ iff $Z(h(x_1), \ldots, h(x_n))$ holds. This representation can be extended in the obvious manner to areas. Relations such as $\gamma \models_h A$, $\gamma \models A$, $A_1 \equiv A_2$, $A_1 \sqsubseteq A_2$, etc, are defined for areas in a similar manner to zones.

For zones Z_1 and Z_2, a renaming from Z_1 to Z_2 is a mapping $\mathcal{R} : Var(Z_1) \mapsto Var(Z_2)$ such that $type(x) = type(\mathcal{R}(x))$. We use $Ren(Z_1)(Z_2)$ to denote the set of renamings from Z_1 to Z_2.

Lemma 12.1. For zones Z_1 and Z_2 with $Var(Z_1) = \{x_1, \ldots, x_m\}$ and $Var(Z_2) = \{y_1, \ldots, y_n\}$, it is the case that $Z_1 \sqsubseteq Z_2$ iff

$$\forall y_1, \ldots, y_n. \left(\begin{array}{c} Z_2(y_1, \ldots, y_n) \implies \\ \\ \bigvee_{\mathcal{R} \in Ren(Z_1)(Z_2)} Z_1^{\odot}(\mathcal{R}(x_1), \ldots, \mathcal{R}(x_m)) \end{array} \right)$$

Notice that the above is a DBL-formula.

13. Experimentation

We have implemented two prototypes to check universality for single-clock timed automata. One of the implementations is based on zones, whereas the other one uses a more compact representation of zones, called Difference Decision Diagrams (DDD), and is based on a package developed at the Technical University of Denmark [31]. We have used these prototypes to check several timed automata for universality. As a reference tool, we used the region-based implementation developed at the Oxford University Computing Laboratory.

In Table 1 we present the results of the tests. For each timed automaton, we give the number of control states, edges, $cmax$, whether universality holds or not, and the execution time for each of the three methods. We use "not term." to indicate that the program did not terminate after more than 24 hours, or that the program stopped without solving the problem due to an out-of-memory exception. All tests were conducted on a Sun workstation with 4.0 GB memory and a 1.0 GHz UltraSPARC-IIIi processor. For both the zone- and region-based implementations we used Java version 1.5.0_05. The DDD-based implementation is compiled with gcc version 2.7.2.3.

In 16 out of 26 tests the execution time of the DDD-based program is smaller than that of the other programs. However, the zone-based prototype is almost as efficient as the DDD-based prototype, as the differences between the execution times are very small, i.e., within a time span of no more than seconds in most of the cases. This is in contrast to the significant differences between the run times of region- and zone-based implementations, varying between milliseconds and hours. As expected, the region-based implementation performs badly for high values of $cmax$. Notice that the run times of both the DDD- and the zone-based prototypes remain relatively stable under changes of the value of $cmax$.

Table 1. Experimental results

$\|S\|$	$\|E\|$	$cmax$	univ?	Region	Zone	DDD
3	4	1	no	21 ms	13 ms	10 ms
3	4	25	no	364 ms	13 ms	0 ms
3	4	50	no	636 ms	14 ms	10 ms
3	4	10000	no	4 hr 49 min 38 sec 601 ms	13 ms	10 ms
10	22	2	yes	639 ms	61 ms	70 ms
10	22	6	yes	550 ms	41 ms	50 ms
10	22	25	yes	1 sec 526 ms	40 ms	70 ms
10	29	135	yes	20 s 981 ms	4 sec 418 ms	not term.
10	29	235	yes	1 min 9 sec 20 ms	3 sec 558 ms	not term.
10	29	335	yes	2 min 24 sec 21 ms	3 sec 746 ms	not term.
10	38	335	yes	1 min 43 sec 175 ms	20 sec 184 ms	not term.
10	44	35	no	3 sec 181 ms	4 min 28 sec 762 ms	1 sec 10 ms
10	44	170	no	27 sec 227 ms	2 min 57 sec 715 ms	670 ms
10	44	560	no	1 min 25 sec 289 ms	6 sec 758 ms	870 ms
10	44	1635	no	41 min 20 sec 623 ms	3 sec 523 ms	320 ms
10	44	2635	no	2 hr 44 sec 135 ms	10 sec 300 ms	1 sec 600 ms
10	44	3635	no	2 hr 1 min 26 sec 921 ms	14 sec 174 ms	1 sec 580 ms
10	44	5635	no	5 hr 21 min 9 sec 24 ms	13 sec 457 ms	1 sec 680 ms
10	44	11635	no	not term.	15 sec 207 ms	1 sec 540 ms
10	30	9335	yes	not term.	3 sec 599 ms	not term.
20	53	4335	yes	not term.	7 sec 061 ms	not term.
25	63	3000	yes	not term.	40 sec 324 ms	not term.
20	53	4335	no	not term.	13 sec 132 ms	12 sec 410 ms
10	30	9880	no	not term.	11 sec 52 ms	300 ms
25	65	10000	no	not term.	1 sec 225 ms	480 ms
25	65	10000	no	not term.	10 min 27 sec 614 ms	2 sec 670 ms

The DDD package does not support any maximum-constant abstraction which causes non-termination in some cases.

14. Conclusions

This paper has investigated both the theoretical and practical complexity of the universality problem for timed automata. It has been known for quite some time that this problem is undecidable in general [10]. The starting point of the current paper was the more recent result of [34] that the universality problem (in fact even the language inclusion problem) over finite timed words is decidable for automata with a single clock.

Our first results concern the theoretical complexity of the one-clock universality problem. Over finite words we showed that the problem was non-primitive recursive, and over infinite words we showed that the problem was undecidable.

Of course, good theoretical complexity is not necessary for a verification technique to be applicable to non-trivial examples (witness the success of software model checking). However, being based on the clock regions construction, the algorithm presented in [34] was impractical for all but the smallest examples. Here we have presented a more practical algorithm for deciding universality of one-clock timed automata, based on zones and Difference Decision Diagrams rather than clock regions. Our results show that this new algorithm outperforms the regions-based implementation by several orders of magnitude.

In future we plan to extend the universality algorithm presented herein to handle language inclusion and to model-check safety properties in Metric Temporal Logic [35]. We would also like to investigate its use as a semi-algorithm for checking language inclusion between arbitrary timed automata.

References

[1] Abdulla, P. A., Čerāns, K., Jonsson, B., Tsay, Y.-K.: Algorithmic Analysis of Programs with Well Quasi-Ordered Domains, *Information and Computation*, **160**, 2000, 109–127.

[2] Abdulla, P. A., Deneux, J., Ouaknine, J., Worrell, J.: Decidability and Complexity Results for Timed Automata via Channel Machines, *Proc. ICALP '05*, 32^{nd} *International Colloquium on Automata, Languages, and Programming*, 3580, 2005.

[3] Abdulla, P. A., Henda, N. B., Delzanno, G., Rezine, A.: Regular Model Checking without Transducers (On Efficient Verification of Parameterized Systems), *Proc. TACAS '07*, 13^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, 4424, Springer Verlag, 2007.

[4] Abdulla, P. A., Jonsson, B.: Undecidable Verification Problems for Programs with Unreliable Channels, *Information and Computation*, **130**(1), 1996, 71–90.

[5] Abdulla, P. A., Jonsson, B.: Verifying Programs with Unreliable Channels, *Information and Computation*, **127**(2), 1996, 91–101.

[6] Abdulla, P. A., Jonsson, B.: Verifying Networks of Timed Processes, *Proc. TACAS '98*, 4^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems* (B. Steffen, Ed.), 1384, 1998.

[7] Abdulla, P. A., Nylén, A.: Better is Better than Well: On Efficient Verification of Infinite-State Systems, *Proc. LICS '00*, 16^{th} *IEEE Int. Symp. on Logic in Computer Science*, 2000.

[8] Abdulla, P. A., Nylén, A.: Timed Petri Nets and BQOs, *Proc. ICATPN'2001: 22nd Int. Conf. on application and theory of Petri nets*, 2075, 2001.

[9] Abdulla, P. A., Ouaknine, J., Quaas, K., Worrell, J.: Zone-Based Universality Analysis for Single-Clock Timed Automata, *Proc. of FSEN 2007, IPM International Symposium on Fundamentals of Software Engineering*, 4767, 2007.

[10] Alur, R., Dill, D.: A Theory of Timed Automata, *Theoretical Computer Science*, **126**, 1994, 183–235.

[11] Alur, R., Feder, T., Henzinger, T.: The Benefits of Relaxing Punctually, *Journal of the ACM*, **43**, 1996, 116–146.

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

[13] Alur, R., Henzinge, T.: Real-Time Logics: Complexity and Expressiveness, *Information and Computation*, **104**(1), 1993, 35–77.

[14] Alur, R., Madhusudan, P.: Decision problems for timed automata: A survey, *4th Intl. School on Formal Methods for Computer, Communication, and Software Systems: Real Time*, 3185, 2004.

[15] Alur, R., Torre, S. L., Madhusudan, P.: Perturbed Timed Automata, *Proc. HSCC 05*, 3414, 2005.

[16] Bouyer, P.: Forward Analysis of Updatable Timed Automata, *Formal Methods in System Design*, **24**(3), 2004, 281–320.

[17] Bouyer, P., Dufourd, C., Fleury, E., Petit, A.: Updatable Timed Automata, *Theoretical Computer Science*, **321**(2-3), 2004, 291–345.

[18] Bouyer, P., Markey, N., Ouaknine, J., Worrell, J.: The Cost of Punctuality, *Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science (LICS'07)*, IEEE Computer Society Press, Wrocław, Poland, July 2007.

[19] Cécé, G., Finkel, A., Iyer, S. P.: Unreliable Channels Are Easier to Verify Than Perfect Channels, *Information and Computation*, **124**(1), 10 January 1996, 20–31.

[20] Cormen, T., Leiserson, C., Rivest, R., Stein, C.: *Introduction to Algorithms*, MIT Press, McGraw-Hill, 2001.

[21] Dill, D.: Timing Assumptions and Verification of Finite-State Concurrent Systems, *Automatic Verifi cation Methods for Finite-State Systems*, 407, 1989.

[22] Finkel, A., Schnoebelen, P.: *Well-Structured Transition Systems Everywhere*, Technical Report LSV-98-4, Ecole Normale Supérieure de Cachan, April 1998.

[23] Henzinger, T., Manna, Z., Pnueli, A.: What Good are Digital Clocks?, *Proc. ICALP '92*, 623, 1992.

[24] Henzinger, T. A., Kopke, P. W., Puri, A., Varaiya, P.: What's Decidable about Hybrid Automata?, *J. Comput. Syst. Sci.*, **57**(1), 1998, 94–124.

[25] Laroussinie, F., Markey, N., Schnoebelen, P.: Model Checking Timed Automata with One or Two Clocks, *Proc. CONCUR 2004*, 15^{th} *Int. Conf. on Concurrency Theory*, 3170, 2004.

[26] Larsen, K., Pettersson, P., Yi, W.: UPPAAL in a Nutshell, *Software Tools for Technology Transfer*, **1**(1-2), 1997.

[27] Lasota, S., Walukiewicz, I.: Alternating Timed Automata, *Proc. FOSSACS*, 3441, 2005.

[28] Lasota, S., Walukiewicz, I.: Alternating Timed Automata, *ACM Trans. on Computational Logic*, 2007, To appear.

[29] Marcone, A.: Foundations of BQO Theory, *Transactions of the American Mathematical Society*, **345**(2), 1994.

[30] Mayr, R.: Undecidable Problems in Unreliable Computations, *Theoretical Informatics (LATIN'2000)*, number 1776 in Lecture Notes in Computer Science, 2000.

[31] Møller, J., Lichtenberg, J.: *Difference Decision Diagrams*, Master Thesis, Department of Information Technology, Technical University of Denmark, Building 344, DK-2800 Lyngby, Denmark, August 1998.

[32] Niebert, P., Mahfoudh, M., Asarin, E., Bozga, M., Jain, N., Maler, O.: Verification of Timed Automata via Satisfiability Checking, *FTRTFT'02, 7th Int. Symp. on Formal Techniques in Real-Time and Fault Tolerant Systems*, 2469, 2002.

[33] Ouaknine, J., Worrell, J.: Universality and Language Inclusion for Open and Closed Timed Automata, *Proc. HSCC 03*, 2623, 2003.

[34] Ouaknine, J., Worrell, J.: On the Language Inclusion Problem for Timed Automata: Closing a Decidability Gap, *Proc. LICS '04*, 20^{th} *IEEE Int. Symp. on Logic in Computer Science*, 2004.

[35] Ouaknine, J., Worrell, J.: On the decidability of Metric Temporal Logic, *Proc. 19th Annual Symp. on Logic in Computer Science (LICS'05)*, IEEE Comp. Soc. Press, 2005.

[36] Schnoebelen, Ph.: Verifying Lossy Channel Systems has Nonprimitive Recursive Complexity, *Information Processing Letters*, **83**(5), 2002, 251–261.

[37] Thomson, B. S., Bruckner, J. B., Bruckner, A. M.: *Elementary Real Analysis*, Prentice Hall, 2001.

[38] Yovine, S.: Kronos: A verification tool for real-time systems, *Journal of Software Tools for Technology Transfer*, **1**(1-2), 1997.

A. Appendix - Proofs of Lemmas

To show Lemma 7.2, we first prove the following lemma.

Lemma A.1. For configurations γ_1, γ_2, and γ_3, if $\gamma_1 \xrightarrow{\delta, a} \gamma_2$ and $\gamma_3 \sqsubseteq \gamma_1$, then there is a γ_4 such that $\gamma_3 \xrightarrow{\delta, a} \gamma_4$ and $\gamma_4 \sqsubseteq \gamma_2$.

Proof:
Since $\gamma_3 \sqsubseteq \gamma_1$ it follows by definition that there is a $\gamma_1' \subseteq \gamma_1$ such that $\gamma_1' \equiv \gamma_3$. Let γ_2' be the (unique) configuration such that $\gamma_1' \xrightarrow{\delta, a} \gamma_2'$. Obviously $\gamma_2' \subseteq \gamma_2$. Since $\gamma_1' \equiv \gamma_3$, it follows from Lemma 7.1 that there is a γ_4 such that $\gamma_3 \xrightarrow{\delta, a} \gamma_4$ and $\gamma_2' \equiv \gamma_4$. By definition it follows that $\gamma_4 \sqsubseteq \gamma_2$. □

Lemma 7.2

Proof:
Since $\Gamma_1 \Longrightarrow \Gamma_2$, we know by definition that $\Gamma_1 \xrightarrow{\delta, a} \Gamma_2$ for some δ and a. Let Γ_4 be the (unique) set of configurations such that $\Gamma_3 \xrightarrow{\delta, a} \Gamma_4$. We show that $\Gamma_4 \sqsubseteq \Gamma_2$. Take any $\gamma_2 \in \Gamma_2$. Since $\Gamma_1 \xrightarrow{\delta, a} \Gamma_2$, there is a $\gamma_1 \in \Gamma_1$ such that $\gamma_1 \xrightarrow{\delta, a} \gamma_2$. Since $\Gamma_3 \sqsubseteq \Gamma_1$ it follows by definition that there is a $\gamma_3 \in \Gamma_3$ such that $\gamma_3 \sqsubseteq \gamma_1$. By Lemma A.1 it follows that there is a γ_4 such that $\gamma_3 \xrightarrow{\delta, a} \gamma_4$ and $\gamma_4 \sqsubseteq \gamma_2$. Since $\Gamma_3 \xrightarrow{\delta, a} \Gamma_4$, $\gamma_3 \in \Gamma_3$, and $\gamma_3 \xrightarrow{\delta, a} \gamma_4$, it follows by definition that $\gamma_4 \in \Gamma_4$. Thus, for an arbitrary $\gamma_2 \in \Gamma_2$, we have found a $\gamma_4 \in \Gamma_4$ with $\gamma_4 \sqsubseteq \gamma_2$. It follows that $\Gamma_4 \sqsubseteq \Gamma_2$. Also, $\Gamma_3 \xrightarrow{\delta, a} \Gamma_4$, means that $\Gamma_3 \Longrightarrow \Gamma_4$, and hence the result. □

Lemma 7.3

Proof:
Let Γ be a set of configurations and $rank(\Gamma) = n$ for some $n : 0 < n < \infty$. Then there is a sequence $\Gamma_0 \Longrightarrow \Gamma_1 \Longrightarrow \cdots \Longrightarrow \Gamma_n$, where $\Gamma_0 = \Gamma$ and Γ_n is not accepting. One can easily see that $rank(\Gamma_1) \leq n - 1$. The result follows immediately since $\Gamma_1 \in (\Gamma \Longrightarrow)$. □

Lemma 7.4

Proof:
(if) Assume that Γ is not universal. Then there is a sequence $\Gamma_0 \Longrightarrow \Gamma_1 \Longrightarrow \cdots \Longrightarrow \Gamma_n$, where $\Gamma_0 = \Gamma$ and Γ_n is not accepting. If $n = 0$ then Γ is not accepting. Otherwise, we know that $rank(\Gamma_1) \leq n - 1$ and hence Γ_1 is not universal. The result follows since $\Gamma_1 \in (\Gamma \Longrightarrow)$.
(only if) If Γ is not accepting then Γ is trivially non-universal ($rank(\Gamma) = 0$). Suppose that there is a $\Gamma' \in Post(\Gamma)$ which is not universal. Then, there is a sequence $\Gamma_0 \Longrightarrow \Gamma_1 \Longrightarrow \Gamma_2 \Longrightarrow \cdots \longrightarrow \Gamma_n$, where $\Gamma_0 = \Gamma'$ and Γ_n is not accepting. It follows that $rank(\Gamma) \leq n + 1$, and hence, Γ is not universal.
□

To prove Lemma 11.2 we need first to prove Lemma A.2, Lemma A.3, Lemma A.4, and Lemma A.5.

Lemma A.2. Consider a stable and consistent zone Z and a configuration γ. Let X, Y be a partitioning of $Var\,(Z)$, and let $h : X \mapsto \gamma$ be a mapping such that $\gamma \models_h \exists Y.Z$. Then there is a configuration γ' and a mapping $h' : Y \mapsto \gamma'$ such that $\gamma \cup \gamma' \models_{h \cup h'} Z$.

Proof:
We show the case when $Y = \{y\}$ is a singleton. The claim then follows by induction on the size of Y. We define ρ to be any number in $\mathbb{R}_+$ satisfying the following properties:

- If $(y - x \sim k) \in Z$ then $\rho \sim h(x) + k$.
- If $(x - y \sim k) \in Z$ then $h(x) - k \sim \rho$.

We show that such a ρ exists. Suppose that ρ does not exist. There are two possible cases each leading to a contradiction as follows.

- There is a $(y - x \sim k) \in Z$ such that $0 \not\sim h(x) + k$. Since Z is stable, we know that $(x^0 - y \leq 0) \in Z$ and hence $(x^0 - x \sim k) \in Z$. Since $\gamma \models_h \exists Y.Z$ it follows that $h(x^0) - h(x) \sim k$. From $h(x^0) = 0$ we get $0 \sim h(x) + k$ which is a contradiction.
- There are $(y - x_1 \sim_1 k_1) \in Z$ and $(x_2 - y \sim_2 k_2) \in Z$ such that $h(x_2) - h(x_1) \not\sim k$ where $(\sim, k) = (\sim_1, k_1) + (\sim_2, k_2)$. Since Z is stable, we know that $(y - x_3 \sim_3 k_3) \in Z$ for some $(\sim_3, k_3) \lhd (\sim, k)$. Since $\gamma \models_h \exists Y.Z$ it follows that $h(y) - h(x_3) \sim_3 k_3$, which is a contradiction.

Now, we define $\gamma' = \{q\}$ where $state\,(q) = type\,(y)$ and $val\,(q) = \rho$. Furthermore, we define $h'(y) = q$. From the definitions it follows that $\gamma \cup \gamma' \models_{h \cup h'} Z$. □

Lemma A.3. Consider a consistent zone Z, a configuration γ, a set $X \subseteq Var\,(Z)$, and be a mapping $h : X \mapsto \gamma$ such that $\gamma \models_h Z$. Let $Z' = (\exists\, Var\,(Z) \setminus X)\,.Z$, $\gamma' = \{s |\ \exists x \in X.\ h(x) = s\}$. Let h' be the restriction of h to X. Then $\gamma' \models_{h'} Z'$.

Proof:
Follows immediately from the definitions. □

Lemma A.4.

$$\bigcup_{Z' \in (Z \oplus a)} [\![Z']\!] \ \subseteq\ \left\{\gamma' |\ \exists \gamma \in [\![Z]\!].\ \gamma \xrightarrow{a}_D \gamma'\right\}$$

Proof:
Suppose that $\gamma' \models Z'$ for some $Z' \in (Z \oplus a)$. We show that there is γ such that $\gamma \models Z$ and $\gamma \xrightarrow{a}_D \gamma'$. Since $Z' \in (Z \oplus a)$, we know that $Z' = \exists\, Var\,(Z)\,.Stabilize(Z_1)$ for some $Z_1 \in (Z \otimes a)$. By definition, there is a mapping $h' : Var\,(Z') \mapsto \gamma'$ such that $\gamma' \models_{h'} Z'$. By Lemma A.2 it follows that there is a γ and a mapping $h : Var\,(Z) \mapsto \gamma$ such that $\gamma \cup \gamma' \models_{h \cup h'} Stabilize(Z_1)$. By Lemma 8.1 we know that $\gamma \cup \gamma' \models_{h \cup h'} Z_1$. By Lemma A.3 it follows that $\gamma \models_h Z$, and hence $\gamma \models Z$. Now, we show that $\gamma \xrightarrow{a}_D \gamma'$.

We do that in two steps:

1. Suppose that $q' \in \gamma'$. We show that there is a $q \in \gamma$ such that $q \xrightarrow{a}_D q'$. We know that $q' = h'(y_{(e,x)})$ for some $(e,x) \in (Z \odot a)$. Let e be of the form $(s, s', \phi, a, R, \phi')$. Since $state\,(q') = type\left(y_{(e,x)}\right)$ it follows that $state\,(q') = s'$. Define $q = h(x)$. It follows that $state\,(q) = s$. There are two cases to consider:

   - If $c \notin R$ then by the definition of $Z \otimes a$, we know that $\left(y_{(e,x)} = x\right) \in Z_1$. Since $\gamma \cup \gamma' \models_{h \cup h'} Z_1$ it follows that $val\,(h(x)) = val\left(h'(y_{(e,x)})\right)$, i.e., $val\,(q) = val\,(q')$. Also, by the definition of $Z \otimes a$, we know that $\phi(x) \in Z_1$, and hence, by Lemma 8.1, $val\,(q)$ satisfies ϕ. In summary, we have shown above that $state\,(q) = s$, $state\,(q') = s'$, $val\,(q)$ satisfies ϕ, and $val\,(q) = val\,(q')$. It follows that $q \xrightarrow{a}_D q'$.
   - If $c \in R$ then by definition of $Z \otimes a$, we know that $\left(y_{(e,x)} = x^0\right) \in Z_1$. Since $\gamma \cup \gamma' \models_{h \cup h'} Z_1$ it follows that $val\,(q') = 0$. Also, by the definition of $Z \otimes a$, we know that $\phi(x) \in Z_1$. By Lemma 8.1 It follows that $val\,(q)$ satisfies ϕ. In summary, we have shown above that $state\,(q) = s$, $state\,(q') = s'$, $val\,(q)$ satisfies ϕ, and $val\,(q) = 0$. It follows that $\gamma \xrightarrow{a}_D \gamma'$.

2. Suppose that there is a $q \in \gamma$ such that $q \xrightarrow{a}_D q'$. We show that $q' \in \gamma'$. Since $\gamma \models_h Z$, we know that $q = h(x)$ for some $x \in Var\,(Z)$. Since $q \xrightarrow{a}_D q'$, we know that there is an edge e be of the form $(s, s', \phi, a, R, \phi')$, such that $state\,(q) = s$, $state\,(q') = s'$, and $val\,(q)$ satisfies ϕ. There are two cases:

   - If $c \notin R$ then we know that $val\,(q') = val\,(q)$. Since $val\,(q)$ satisfies ϕ, it follows by the definition of $Z \otimes a$, that $\left(y_{(e,x)} = x\right) \in Z_1$. Lemma 8.1 implies that $val\left(h'\left(y_{(e,x)}\right)\right) = val\,(h(x))$, and hence $val\left(h'\left(y_{(e,x)}\right)\right) = val\,(q) = val\,(q')$. Furthermore, we know that $type\left(y_{(e,x)}\right) = s' = state\,(q')$. The fact that $val\left(h'\left(y_{(e,x)}\right)\right) = val\,(q')$ and $type\left(y_{(e,x)}\right) = state\,(q')$ implies $q' \in \gamma'$.
   - If $c \in R$ then we know that $val\,(q') = 0$. Since $val\,(q)$ satisfies ϕ, it follows by the definition of $Z \otimes a$, that $\left(y_{(e,x)} = x^0\right) \in Z_1$. Lemma 8.1 implies that $val\left(h'\left(y_{(e,x)}\right)\right) = 0 = val\,(q')$. Furthermore, we know that $type\left(y_{(e,x)}\right) = s' = state\,(q')$. The fact that $val\left(h'\left(y_{(e,x)}\right)\right) = val\,(s')$ and $type\left(y_{(e,x)}\right) = state\,(q')$ implies $q' \in \gamma'$.

□

Lemma A.5.

$$\left\{\gamma' \mid \exists \gamma \in [\![Z]\!].\ \gamma \xrightarrow{a}_D \gamma'\right\} \subseteq \bigcup_{Z' \in (Z \oplus a)} [\![Z']\!]$$

Proof:
Suppose that $\gamma \models Z$ and $\gamma \xrightarrow{a}_D \gamma'$. We show that $\gamma' \models Z'$ for some $Z' \in Z \oplus a$. Since $\gamma \models Z$, we know that $\gamma \models_h Z$ for some mapping $h : Var\,(Z) \mapsto \gamma$. For a global state $q = (s, \nu) \in \gamma$, and an edge $e = (s, s', \phi, a, R, \phi')$, we define q_e as follows:

- If ν satisfies ϕ and $c \notin R$ then $q_e = (s', \nu)$.
- If ν satisfies ϕ and $c \in R$ then $q_e = (s', 0)$.
- If ν does not satisfy ϕ then q_e is undefined.

By definition, γ' consists of all q_e which are defined. Consider the zone

$$Z_1 = \left(\bigwedge_{(e,x)\in(Z\odot a)} \phi_{(e,x)} \right) \wedge Z$$

in $Z \otimes a$, where $\phi_{(e,x)}$, with $e = (s, s', \phi, a, R, \phi')$, is defined as follows

- If ν satisfies ϕ and $c \notin R$ then $\phi_{(e,x)} = (y_{(e,x)} = x) \wedge \phi(x)$.
- If ν satisfies ϕ and $c \in R$ then $\phi_{(e,x)} = (y_{(e,x)} = 0) \wedge \phi(x)$.
- If ν does not satisfy ϕ then the corresponding formula $\phi_{(e,x)}$ is missing.

Define the mapping $h' : Var(Z') \mapsto \gamma'$ such that $h'(y_{(e,x)}) = q_e$ if q_e is defined (or equivalently if $\phi_{(e,x)}$ is not missing in Z'), and if $h(x) = q$. By definition it follows that $\gamma \cup \gamma' \models_{h\cup h'} Z_1$. By Lemma 8.1 it follows that $\gamma \cup \gamma' \models_{h\cup h'} Stabilize(Z_1)$. Lemma A.3 implies that $\gamma' \models \exists Var(Z).Stabilize(Z_1)$. By definition $(\exists Var(Z).Stabilize(Z_1)) \in Z \oplus a$. □

Lemma 11.2

Proof:
Follows directly from Lemma A.4 and Lemma A.5. □

To show Lemma 12.1, we show Lemma A.6, Lemma A.7, Lemma A.8, and Lemma A.9.

Lemma A.6. For a consistent and stable zone Z, it is the case that $Z^{\odot} \equiv Z$.

Proof:
Let $Z = \varphi_1 \wedge \cdots \wedge \varphi_n$, i.e., $Z^{\odot} = {\varphi_1}^{\odot} \wedge \cdots \wedge {\varphi_n}^{\odot}$. Suppose that $\gamma \models_h Z^{\odot}$. We show that $\gamma' \models Z$ for some $\gamma' \equiv \gamma$. Without loss of generality, we assume that $Var(Z)$ is of the form $\{x_1, \ldots, x_\ell, x_{\ell+1}, \ldots, x_m\}$, where the following two conditions are satisfied:

1. $h(x_i) \leq cmax$ for each $i : 1 \leq i \leq \ell$; and
2. If $(x_j - x_i \sim k) \in Z$ for some $\ell < i, j \leq m$, and $(\sim, k) \triangleleft (\leq, -1)$, then $j < i$.

The second condition can be satisfied since Z is consistent.

We derive a sequence $\gamma_\ell, \gamma_{\ell+1}, \ldots, \gamma_m$ of configurations and a corresponding sequence $h_\ell, h_{\ell+1}, \ldots, h_m$ such that $\gamma_i \models_{h_i} \exists x_{i+1}, \ldots, x_m.Z$ for all $i : \ell \leq i \leq m$.

We define $h_\ell(x_i) = h(x_i)$ for each $i : 1 \leq i \leq \ell$; and define $\gamma_\ell = \{h(x_i) | \ 1 \leq i \leq \ell\}$. Suppose that $\gamma_\ell \not\models_{h_\ell} \exists x_{\ell+1}, \ldots, x_m.Z$. It follows that $h_\ell(x_j) - h_\ell(x_i) \not\sim k$ for some $(x_j - x_i \sim k) \in Z$ where $1 \leq i, j \leq \ell$. There are two possible cases depending on whether k is negative. We show that we get a contradiction in each case:

- $k \geq 0$ and $(x_j - x_i \sim k) \vee (x_i > cmax - k) \in Z^{\odot}$. Since $\gamma \models_h Z^{\odot}$, we have that $(h(x_j) - h(x_i) \sim k) \vee (h(x_i) > cmax - k)$. Since $h_\ell(x_j) = h(x_j)$ and $h_\ell(x_i) = h(x_i)$ it follows that $h(x_j) - h(x_i) \not\sim k$. This means that $h(x_i) + k \sim h(x_j)$ and $h(x_i) > cmax - k$, and hence $h(x_j) > cmax$ which contradicts condition 1 above.

- $k < 0$ and $(x_j - x_i \sim k) \vee (x_i > cmax) \in Z^{\odot}$. Since $\gamma \models_h Z^{\odot}$, we have that $(h(x_j) - h(x_i) \sim k) \vee (h(x_i) > cmax)$. Since $h_\ell(x_j) = h(x_j)$ and $h_\ell(x_i) = h(x_i)$ it follows that $h(x_j) - h(x_i) \not\sim k$. This means that $h(x_i) > cmax$ which contradicts condition 1 above.

Now, we consider $i : \ell < i \leq m$. We define $h_i(x_j) = h_{i-1}(x_j)$ if $j < i$, and define $h_i(x_i) = \rho$, where ρ is any number in $\mathbb{R}_+$ satisfying the following properties:

(a) $cmax < \rho$.

(b) If $(x_j - x_i \sim k) \in Z$ for some $j < i$ then $h_i(x_j) - k \sim \rho$.

(c) If $(x_i - x_j \sim k) \in Z$ for some $j < i$ then $\rho \sim h_i(x_j) + k$.

We show that such a ρ exists. Suppose that ρ does not exist. There are two possible cases each leading to a contradiction as follows.

- Conditions (b) and (c) cannot be satisfied. This means that there are $j_1, j_2 : 1 \leq j_1, j_2 < i$ such that $(x_{j_1} - x_i \sim_1 k_1) \in Z$, $(x_i - x_{j_2} \sim_2 k_2) \in Z$, and $h_i(x_{j_1}) - h_i(x_{j_2}) \not\sim_3 k_3$ where $(\sim_3, k_3) = (\sim_1, k_1) + (\sim_2, k_2)$. Since Z is consistent, we know that $(x_{j_1} - x_{j_2} \sim_4 k_4) \in Z$ for some $(\sim_4, k_4) \triangleleft (\sim_3, k_3)$. Notice that $(x_{j_1} - x_{j_2} \sim_4 k_4) \in (\exists x_i, \ldots, x_m.Z)$. Since $\gamma_{i-1} \models_{h_i} \exists x_i, \ldots, x_m.Z$ it follows that $h_{i-1}(x_{j_1}) - h_{i-1}(x_{j_2}) \sim_4 k_4$. From $h_i(x_{j_1}) = h_{i-1}(x_{j_1})$, $h_i(x_{j_2}) = h_{i-1}(x_{j_2})$, it follows that $h_i(x_{j_1}) - h_i(x_{j_2}) \sim_4 k_4$ which is a contradiction.

- Conditions (a) and (c) cannot be satisfied. This means that $(x_i - x_j \sim k) \in Z$, for some $1 \leq j < i$ and $h_i(x_j) \leq cmax - k$. We distinguish two subcases
  - If $1 \leq j \leq \ell$. Again, we distinguish two subcases:
    - If $k < 0$ then $((x_i - x_j \sim k) \vee (x_j > cmax)) \in Z^{\odot}$. Since $\gamma \models_h Z^{\odot}$ it follows that $(h(x_i) - h(x_j) \sim k) \vee (h(x_j) > cmax)$. By condition 1, we know that $h(x_j) \leq cmax$. This means that $h(x_i) - h(x_j) \sim k$. By condition 1, we know that $h(x_i) > cmax$ and hence $cmax < h(x_j) + k$. Since $h_i(x_j) = h(x_j)$ it follows that $cmax < h_i(x_j) + k$ which contradicts $h_i(x_j) \leq cmax - k$.
    - If $k \geq 0$ then $((x_i - x_j \sim k) \vee (x_j > cmax) - k) \in Z^{\odot}$. Since $\gamma \models_h Z^{\odot}$ it follows that $(h(x_i) - h(x_j) \sim k) \vee (h(x_j) > cmax) - k$. If $h(x_i) - h(x_j) \sim k$ then we get a contradiction in the same manner as above. Otherwise, $h(x_j) > cmax - k$ which contradicts $h_i(x_j) \leq cmax - k$.
  - If $\ell < j \leq i - 1$. If $k < 0$, then condition 2 implies that $i < j$ which contradicts $i < j$. This means that $0 \leq k$. Since $h_i(x_j) > cmax$ it follows that $h_i(x_j) + k > cmax$ which contradicts $h_i(x_j) \leq cmax - k$.

It remains to show that $\gamma_m \equiv \gamma$. This follows from the fact that $h_m(x_i) = h(x_i)$ for all $i : 1 \leq i \leq \ell$ and both $h_m(x_i) > cmax$ and $h(x_i) > cmax$ for all $i : \ell < i \leq m$. □

Lemma A.7. Consider an area A and configurations γ_1 and γ_2. If $\gamma_1 \models A$ and $\gamma_1 \equiv \gamma_2$ then $\gamma_2 \models A$.

Proof:
The proof follows from the definitions. □

Lemma A.8. For zones Z_1 and Z_2 with $Var\,(Z_1) = \{x_1, \ldots, x_m\}$ and $Var\,(Z_2) = \{y_1, \ldots, y_n\}$, if

$$\forall y_1, \ldots, y_n. \left(\begin{array}{c} Z_2(y_1, \ldots, y_n) \implies \\ \\ \bigvee_{\mathcal{R} \in Ren(Z_1)(Z_2)} {Z_1}^{\odot}(\mathcal{R}(x_1), \ldots, \mathcal{R}(x_m)) \end{array} \right)$$

then $Z_1 \sqsubseteq Z_2$.

Proof:
Suppose that $\gamma_2 \models Z_2$. We show that there is a γ_3 such that $\gamma_3 \models Z_1$ and $\gamma_3 \sqsubseteq \gamma_2$. Since $\gamma_2 \models Z_2$, we know that there is a mapping $h_2 : Var\,(Z_2) \mapsto \gamma_2$ such that $\gamma_2 \models_{h_2} Z_2$. This means that $Z_2(val\,(h_2(y_1)), \ldots, val\,(h_2(y_n)))$ holds. By the premise of the lemma it follows that there is a renaming $\mathcal{R} \in Ren(Z_1)(Z_2)$ such that ${Z_1}^{\odot}(val\,(h_2(\mathcal{R}(x_1))), \ldots, val\,(h_2(\mathcal{R}(x_m))))$ holds. Define $\gamma_1 = \{q |\ \exists x \in Var\,({Z_1}^{\odot})\,.\, h_2(\mathcal{R}(x)) = q\}$. Clearly $\gamma_1 \subseteq \gamma_2$. Define the mapping $h_1 : Var\,({Z_1}^{\odot}) \mapsto \gamma_1$ such that $h_1(x) = h_2(\mathcal{R}(x))$. We show that $\gamma_1 \models_{h_1} {Z_1}^{\odot}$:

- For each $x \in Var\,({Z_1}^{\odot})$, we have $type\,(x) = type\,(\mathcal{R}(x)) = state\,(h_2(\mathcal{R}(x))) = state\,(h_1(x))$.
- Suppose that $(y - x \sim k) \vee (x > cmax - k) \in {Z_1}^{\odot}$. We show that $(val\,(h_1(y)) - val\,(h_1(x)) \sim k) \vee (val\,(h_1(x)) > cmax - k)$ holds. Since ${Z_1}^{\odot}(val\,(h_2(\mathcal{R}(x_1))), \ldots, val\,(h_2(\mathcal{R}(x_m))))$ holds we know that $(val\,(h_2(\mathcal{R}(y))) - val\,(h_2(\mathcal{R}(x))) \sim k) \vee (val\,(h_2(\mathcal{R}(x))) > cmax - k)$ holds. Since $h_1(x) = h_2(\mathcal{R}(x))$, it follows that $(val\,(h_1(y)) - val\,(h_1(x)) \sim k) \vee (val\,(h_1(x)) > cmax - k)$ holds.

By Lemma A.6, there is a γ_3 such that $\gamma_3 \equiv \gamma_1$ and $\gamma_3 \models Z_1$. Since $\gamma_1 \subseteq \gamma_2$, we have $\gamma_3 \sqsubseteq \gamma_2$. □

Lemma A.9. For zones Z_1 and Z_2 with $Var\,(Z_1) = \{x_1, \ldots, x_m\}$ and $Var\,(Z_2) = \{y_1, \ldots, y_n\}$, if $Z_1 \sqsubseteq Z_2$ then

$$\forall y_1, \ldots, y_n. \left(\begin{array}{c} Z_2(y_1, \ldots, y_n) \implies \\ \\ \bigvee_{\mathcal{R} \in Ren(Z_1)(Z_2)} {Z_1}^{\odot}(\mathcal{R}(x_1), \ldots, \mathcal{R}(x_m)) \end{array} \right)$$

Proof:
Suppose that there is a mapping $g : \{y_1, \ldots, y_n\} \mapsto \mathbb{R}_+$ such that $Z_2(g(y_1), \ldots, g(y_n))$ holds. We show that there is a renaming $\mathcal{R}$ from Z_1 to Z_2 such that ${Z_1}^{\odot}(g(\mathcal{R}(x_1)), \ldots, g(\mathcal{R}(x_m)))$ holds. Define $\gamma_2 = \{q_1, \ldots, q_n\}$ where $state\,(q_i) = type\,(y_i)$ and $val\,(q_i) = g(y_i)$. Obviously, $\gamma_2 \models_{h_2} Z_2$, where $h_2(y_i) = q_i$ for each $i : 1 \leq i \leq n$. Since $Z_1 \sqsubseteq Z_2$, there is a γ_1 such that $\gamma_1 \sqsubseteq \gamma_2$ and $\gamma_1 \models Z_1$. It follows by definition that there is a γ_3 such that $\gamma_3 \subseteq \gamma_2$ and $\gamma_1 \equiv \gamma_3$. Since $\gamma_1 \models Z_1$ it follows by Lemma A.6 that there is a γ_4 such that $\gamma_4 \equiv \gamma_1$ and $\gamma_4 \models {Z_1}^{\odot}$. From $\gamma_1 \equiv \gamma_3$, it follows that $\gamma_4 \equiv \gamma_3$ and hence by Lemma A.7 we get $\gamma_3 \models {Z_1}^{\odot}$, i.e., there is a mapping $h_1 : Var\,(Z_1) \mapsto \gamma_3$ such that $\gamma_3 \models_{h_1} {Z_1}^{\odot}$. This means that ${Z_1}^{\odot}(val\,(h_1(x_1)), \ldots, val\,(h_1(x_m)))$ holds. Define the renaming $\mathcal{R} \in Ren(Z_1)(Z_2)$ such that $\mathcal{R}(x) = y$ if $g(y) = val\,(h_1(x))$. This implies that ${Z_1}^{\odot}(g(\mathcal{R}(x_1)), \ldots, g(\mathcal{R}(x_m)))$ holds. □

Lemma 12.1

Proof:
Follows directly from Lemma A.8 and Lemma A.9. □