Zone-Based Universality Analysis for Single-Clock Timed Automata

Parosh Aziz Abdulla[1], Joël Ouaknine[2], Karin Quaas[1], and James Worrell[2]

[1] Department of Computer Systems, Uppsala University, Sweden
{parosh,karin}@it.uu.se
[2] Oxford University Computing Laboratory, UK
{joel,jbw}@comlab.ox.ac.uk

Abstract. During the last years, timed automata have become a popular model for describing the behaviour of real-time systems. In particular, there has been much research on problems such as language inclusion and universality. It is well-known that the universality problem is undecidable for the class of timed automata with two or more clocks. Recently, it was shown that the problem becomes decidable if the automata are restricted to operate on a single clock variable. However, existing algorithms use a region-based constraint system and suffer from constraint explosion even for small examples. In this paper, 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 based on our method, and checked universality for a number of timed automata. Comparisons with a region-based prototype 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 [1], timed automata have been used as the foundation for several verification algorithms and tools (see [2] 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? (i.e., 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 [3,4,2,5,6,7].

In particular, the paper [8] shows that both the universality and language inclusion problems are decidable for the class of timed automata which are restricted to operate on a single clock. The paper uses a variant of *regions* as a

F. Arbab and M. Sirjani (Eds.): FSEN 2007, LNCS 4767, pp. 98–112, 2007.

symbolic representation for sets of states in the universality algorithm; and uses the theory of *well quasi-orderings*, for proving termination of the algorithm. Despite the positive result in [8], deriving an algorithm which has a reasonable efficiency is still a difficult challenge. In fact, it is shown in [9] that the universality problem has a non-primitive recursive complexity for single-clock timed automata. In addition, it is well-known that the region representation is in general very inefficient and tends to explode even on very small examples.

In this paper, we propose a new formalism 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 [10] and timed Petri nets [11]. 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 [12] and UPPAAL [13].

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

Outline. In the next section, we give some preliminaries of timed automata. In Section 3 we consider the universality problem for configurations. In Section 4, we introduce zones, and in Section 5, we describe the zone-based universality algorithm. In Section 6, we show that the algorithm is guaranteed to terminate. We devote Section 7 and Section 8 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 9, we report some experimental results. Finally, we give some conclusions and directions for future work in Section 10.

2 Timed Automata

In this section, we recall the basic definitions for timed automata, and concentrate on the class where the automata operate on a single clock.

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

Timed Words. Let Σ be a finite alphabet. 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 sequence $t = (t_0, a_0)(t_1, a_1)(t_2, a_2) \dots (t_n, a_n)$ of timed events whose sequence of timestamps $t_0 t_1 t_2 \dots t_n$ is non-decreasing. We write $T\Sigma^*$ for the set of finite timed words over the alphabet Σ.

Timed Automata. We consider timed automata which operate on a single clock (in the sequel referred to as clock c). We define the set Φ of clock constraints to be conjunctions of formulas of the form $c \sim k$, $k \sim c$, where $k \in \mathbb{N}$ 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,
- $E \subseteq S \times S \times \Phi \times \Sigma \times \{tt, ff\}$ is a finite set of edges. An edge (s, s', ϕ, a, R) allows an a-labelled transition from s to s', provided that the precondition ϕ on clock c is met. Afterwards, c is either reset to 0 (if $R = tt$), or its value remains unchanged (if $R = ff$).

We let *cmax* be the maximum natural number which appears on the edges of the automaton. A *global state* q of $\mathcal{A}$ is a pair (s, ν), where $s \in S$ is a control state and $\nu \in \mathbb{R}_+$ represents the value of clock c. We use $\mathit{state}\,(q)$ and $\mathit{val}\,(q)$ to denote s and ν. We say that q is *accepting* if $\mathit{state}\,(q) \in F$. The *initial global state* q_{init} is of the form $(s_{init}, 0)$.

Transition System. We define a transition relation on global states. For a global state q, we let $q + \delta$ be the global state q' such that $\mathit{state}\,(q') = \mathit{state}\,(q)$ and $\mathit{val}\,(q') = \mathit{val}\,(q) + \delta$. 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', ϕ, a, R) in E and the following conditions are satisfied: (i) ν satisfies ϕ; (ii) $\nu' = 0$ if $R = tt$; and (iii) $\nu' = \nu$ if $R = ff$. 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* (of $\mathcal{A}$) from q is a finite sequence of transitions

$$q_0 \xrightarrow{\delta_0, a_0} q_1 \xrightarrow{\delta_1, a_1} q_2 \xrightarrow{\delta_2, a_2} \cdots \xrightarrow{\delta_{n-1}, a_{n-1}} q_n \tag{1}$$

where $q_0 = q$. The run is *accepting* if q_n is accepting. We use $L(q)$ to denote the set of timed words of the form $(t_0, a_0)(t_1, a_1) \dots (t_{n-1}, a_{n-1})$ such that there is an accepting run from q of the above form and $t_j = \sum_{i=0}^{j} \delta_i$ for each $j : 0 \leq j < n$.

We say that q is *universal* if $L(q) = T\Sigma^*$. In the *universality problem*, we are given an automaton, and are asked whether the initial global state is universal or not.

3 Configurations

To solve the universality problem for global states, we study a more general problem, namely the universality problem for (sets of) *configurations*.

Configurations. A *configuration* γ is a finite set of global states. A configuration 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 $\gamma' = \left\{q' \middle| \exists q \in \gamma.\ q \xrightarrow{\delta}_T q'\right\}$. 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

$$\gamma_0 \xrightarrow{\delta_0, a_0} \gamma_1 \xrightarrow{\delta_1, a_1} \gamma_2 \xrightarrow{\delta_2, a_2} \cdots \xrightarrow{\delta_{n-1}, a_{n-1}} \gamma_n \tag{2}$$

where $\gamma_0 = \gamma$. The run is *accepting* if γ_n is accepting. We define $L(\gamma)$ in a similar manner to the case of global states. We say that γ is *universal* if $L(\gamma) = T\Sigma^*$. Notice that $L(\gamma) = \bigcup_{q \in \gamma} L(q)$.

Sets of Configurations. 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 $\Gamma' = \left\{\gamma' \middle| \exists \gamma \in \Gamma.\ \gamma \xrightarrow{\delta}_T \gamma'\right\}$. The definitions of the other transition relations are extended analogously. Also the notions of a run, an accepting run, $L(\Gamma)$, and universality, are extended in a similar manner to the case of sets of configurations.

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 $q, q_1, q_2 \in \gamma$:

- $\mathit{state}\,(q) = \mathit{state}\,(h(q))$.
- $\mathit{val}\,(q) \leq \mathit{cmax}$ iff $\mathit{val}\,(h(q)) \leq \mathit{cmax}$.
- if $\mathit{val}\,(q) \leq \mathit{cmax}$ then $\lfloor \mathit{val}\,(q) \rfloor = \lfloor \mathit{val}\,(h(q)) \rfloor$.
- if $\mathit{val}\,(q) \leq \mathit{cmax}$ then $\mathit{fract}\,(\mathit{val}\,(q)) = 0$ iff $\mathit{fract}\,(\mathit{val}\,(h(q))) = 0$.
- if $\mathit{val}\,(q_1) \leq \mathit{cmax}$ and $\mathit{val}\,(q_2) \leq \mathit{cmax}$ then $\mathit{fract}\,(\mathit{val}\,(q_1)) \leq \mathit{fract}\,(\mathit{val}\,(q_2))$ iff $\mathit{fract}\,(\mathit{val}\,(h(q_1))) \leq \mathit{fract}\,(\mathit{val}\,(h(q_2)))$.

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 [1].

Lemma 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 both $\Gamma \sqsubseteq \Gamma'$ and $\Gamma' \sqsubseteq \Gamma$. The following lemma follows from Lemma 1.

Lemma 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 *distance* $dist(\Gamma)$ 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, $dist(\Gamma)$ gives the shortest distance through $\Longrightarrow$ from Γ to a non-accepting set of configurations. If Γ is universal then we define $dist(\Gamma) = \infty$. Notice that $dist(\Gamma) = 0$ 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 3. *For a set Γ of configurations, if $0 < dist(\Gamma) < \infty$ then there is a $\Gamma' \in (\Gamma \Longrightarrow)$ such that $dist(\Gamma') < dist(\Gamma)$.*

Lemma 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 2, implies the following lemma. The lemma shows the relation between the entailment relation and the distance function.

Lemma 5. *For sets Γ and Γ' of configurations, if $\Gamma \sqsubseteq \Gamma'$ then $dist(\Gamma) \leq dist(\Gamma')$.*

4 Zones

We will use zones as a symbolic representation of (infinite) sets of configurations in our universality algorithm. We assume a timed automaton $\mathcal{A} = (\Sigma, S, s_{init}, F, E)$. For each $s \in S$, we will use a set X^s of variables ranging over $\mathbb{R}_+$. We use X to denote the set $\bigcup_{s \in S} X^s$. For $x \in X^s$, we use $type\,(x)$ to denote the control state s.

Zones. A *zone condition* φ is one of the forms $x \sim k$, $k \sim x$, or $y - x \sim k$, where $\sim \in \{\leq, <\}$, $x, y \in X$, and k is an integer. A *zone* Z is a finite conjunction of zone conditions. Sometimes, we consider a zone Z to be a set and write, for instance, $(x \sim k) \in Z$ to indicate that $x \sim k$ is one of the conjuncts in Z. We use $Var\,(Z)$ to denote the set of variables which occur in Z.

Consider a zone Z, a configuration γ, and a mapping $h : Var(Z) \mapsto \gamma$. We write $\gamma \models_h Z$ to denote that, for each $x, y \in Var(Z)$, the following conditions are satisfied:

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

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.

We say that Z is *universal* if $[\![Z]\!]$ is universal. Similarly, we say that Z is *accepting* if $[\![Z]\!]$ is accepting. Let Y be a set of variables. By $Z[Y]$, 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 $\mathcal{Z}[Y]$ to denote the set $\{Z[Y] \mid Z \in \mathcal{Z}\}$.

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 $dist(Z)$ to denote $dist([\![Z]\!])$. 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 7, we show that such a set exists and is computable. A zone Z is said to be *consistent* if $[\![Z]\!] \neq \emptyset$.

Lemma 6. *For a zone Z, we can check whether Z is consistent or not.*

Notice that an inconsistent zone is trivially universal.

Normal Form. A zone Z is said to be in *stable* if the following four conditions are satisfied:

- If $(y - x \leq k_1) \in Z$ and $(z - y \leq k_2) \in Z$ then $(z - x \leq k_3) \in Z$ for some $k_3 \leq k_1 + k_2$.
- If $(x \leq k_1) \in Z$ and $(y - x \leq k_2) \in Z$ then $(y \leq k_3) \in Z$ for some $k_3 \leq k_1 + k_2$.
- If $(y - x \leq k_1) \in Z$ and $(k_2 \leq y) \in Z$ then $(k_3 \leq x) \in Z$ for some $k_3 \geq k_2 - k_1$.
- If $(y - x \leq k_1) \in Z$ then $(k_2 \leq x) \in Z$ for some $k_2 \geq -k_1$.
- Similar conditions hold in the case of strict inequalities.

A zone Z is said to be in *normal form* if all the constants appearing in the definition of Z are less than or equal to $cmax$. It is straightforward [15] to show the following

Lemma 7. *For each zone Z, we can construct:*

- *a stable zone Z_S such that (i) $Var(Z_S) = Var(Z)$; and (ii) $\gamma \models_h Z_S$ iff $\gamma \models_h Z$ for each γ and h*
- *A zone Z_N in normal form such that $Z_N \equiv Z$.*

Notice that the first part of the lemma implies that $[\![Z_S]\!] = [\![Z]\!]$. We use $Stabilize(Z)$ and $Norm(Z)$ to denote Z_S and Z_N respectively. For a set $\mathcal{Z}$ of zones, we define $Stabilize(\mathcal{Z}) = \{Stabilize(Z) |\ Z \in \mathcal{Z}$ and Z is consistent$\}$.

5 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 ⋃ {Norm(Z')| Z' ∈ 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 which have not yet been analyzed; and a set `Explored`, initialized to the empty set, of zones which contains information about the set of zones which 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`. $dist$(Z) $<$ $dist$(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 5. If Z_{init} is non-universal, then the second invariant and Lemma 5 imply that there is still some $Z'' \in$ `ToExplore`

such that $dist(Z'') < dist(Z') \leq dist(Z)$. This means that the second invariant will also be preserved by this step.
- Otherwise, we generate the zones in $Post(Z)$, normalize each one of them (Lemma 7), and then put it in `ToExplore`. The first invariant will be preserved by Lemma 4, while the second invariant will be preserved by Lemma 7, Lemma 5, and Lemma 3.

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

- The algorithm terminates (done in Section 6).
- We can compute *Post* and can check the entailment relation $\sqsubseteq$ on zones (done in Section 7 and Section 8).

Remark. Observe that the correctness of the algorithm is preserved in case we replace $\sqsubseteq$ in the algorithm by any ordering $\sqsubseteq'$ such that $\sqsubseteq' \subseteq \sqsubseteq$ (i.e., $Z \sqsubseteq' Z'$ implies $Z \sqsubseteq Z'$).

6 Termination

Using the methodology of [16] it can be shown that the universality algorithm of Section 5 is guaranteed to terminate in case $\sqsubseteq$ is a *well quasi-ordering (WQO)*. Following the framework of [14], we show that $\sqsubseteq$ in fact satisfies a stronger property than WQO; namely that it is a *better quasi-ordering (BQO)*.

6.1 WQOs and BQOs

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 a set. Notice that if $\preceq$ is a WQO then $\min(B)$ is finite.

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 monotone injection h from $\{1, \ldots, m\}$ to $\{1, \ldots, n\}$ such that $x_i \preceq y_{h(i)}$ for each $i : 1 \leq i \leq m$. 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 8. *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 BQOs[1] [14,17].

Lemma 9. *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* $sign\,(q)$ 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 a word over $S \times \{0, 1, \ldots, 2 \cdot cmax + 1\}$ 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[2] 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 see [18] or [8]).

Lemma 10. *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 9 (Property 2 and Property 3) gives the following:

Lemma 11. *$\preceq$ is a BQO on the set of configurations.*

From Lemma 10, Lemma 11, and Lemma 9 (Property 5) we get the following:

[1] The technical definition of BQOs is quite complicated and can be found in e.g. [14]. The actual definition is not needed for understanding the rest of the paper, and is therefore omitted here.

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

Corollary 1. $\sqsubseteq$ *is a BQO on the set of configurations.*

From the definition of $\sqsubseteq$ on zones, Corollary 1, Lemma 8, and Lemma 9 (Property 4) we get the following

Lemma 12. $\sqsubseteq$ *is a BQO on zones.*

Lemma 12 and Lemma 9 (Property 1) give the following:

Corollary 2. $\sqsubseteq$ *is a WQO on zones.*

7 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 $[\![Z]\!] \xrightarrow{\delta}_T [\![Z']\!]$. 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 7). We can derive $Post_T(Z)$ from Z'' by deleting all clock conditions of the forms $x \leq k$ and $x < k$ in Z''. This gives the following:

Lemma 13. *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 in $x \in Var(Z)$ 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 $e = (s, s', \phi, a, R)$ 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 $e = (s, s', \phi, a, R)$, the set $Z \odot a$ contains all pairs (e, x) which are matching, i.e., $type(x) = s$. Each such a 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 $R = \mathit{ff}$ then $Z \otimes (e,x) = \{(y_{(e,x)} = x) \wedge \phi(x) \ , \ \neg\phi(x)\}$.
– if $R = \mathit{tt}$ then $Z \otimes (e,x) = \{(y_{(e,x)} = 0) \wedge \phi(x) \ , \ \neg\phi(x)\}$.

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 $\mathit{Post}(Z)$; or (ii) ϕ is not satisfied in which case no new variable is added to $\mathit{Post}(Z)$. If a new global state is added then, depending on the value of R, there are two possibilities: either (i) its clock value is equal to the clock value of the original global state; or (ii) 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 $y_{(e,x)} = 0$.

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:

$$Z \oplus a = \ (\mathit{Stabilize}\,(Z \otimes a))\,[\mathit{Var}\,(Z)]$$

Each member of $Z \otimes a$ is a zone which represents the conjunction of the original zone Z with one of the zones in $\mathit{Post}_D(Z)$. 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 $\mathit{Var}\,(Z)$. The following lemma shows correctness of the above construction.

Lemma 14. $\mathit{Post}_D(Z) = \bigcup_{a \in \Sigma} Z \oplus a.$

8 Checking Entailment

In this section, we describe how to implement the entailment relation $\sqsubseteq$ on zones. In fact, there are two methods of computing $Z_1 \sqsubseteq Z_2$. The first method is to generate the regions in Z_1 and Z_2, and compare them for entailment. This method is still less sensitive to constraint explosion than regions-based methods (methods which only use regions), since only a subset of the regions (namely the ones in Z_1 and Z_2) need to be stored at a time. Another method (which we have used in our experimentation) is to construct a logical formula (in a decidable theory) which gives a characterization of the entailment relation. More precisely, the formula corresponds to an ordering $\sqsubseteq'$ on zones which implies $\sqsubseteq$. As indicated in Section 5, the correctness of the universality algorithm will be preserved using the new ordering $\sqsubseteq'$. However, the algorithm will not be guaranteed to terminate unless $\sqsubseteq'$ itself is a WQO. This is due to the fact that zones may avoid being discarded although they are entailed (according to $\sqsubseteq$) by other zones. The use of $\sqsubseteq'$ may still be motivated if they run efficiently on more examples in practice.

Here, we use formulas in a decidable logic which we call *Difference Bound Logic (DBL)*. The atomic formulas are either of the form $x \leq k$ or 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. It is easy to see that validity of DBL-formulas is NP-complete.

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 $\mathbb{N}^m$. Observe that $\gamma \models_h Z$ iff $Z(h(x_1), \ldots, h(x_n))$ holds. 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 15. *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*

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

then $Z_1 \sqsubseteq Z_2$.

Notice that the above is a DBL-formula.

Remark. Lemma 15 defines an ordering $\sqsubseteq'$ which implies $\sqsubseteq$. More precisely, in $\sqsubseteq'$ we take into consideration clock differences even for clocks whose values are greater than *cmax*. In fact, we can modify $\sqsubseteq'$ so that it coincides with $\sqsubseteq$. This can be achieved by modifying the disjunction through adding formulas which equate clock values greater than *cmax*. This can be expressed as a DBL-formula in a straightforward manner. In this manner, the termination of the algorithm will still be guaranteed when using $\sqsubseteq'$.

9 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 [19]. 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.

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

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 $cmax$.

10 Conclusions and Future Work

We have presented a zone-based algorithm for solving the universality problem for timed automata with a single clock. We prove termination of the algorithm using the theory of better quasi-orderings, a refinement of the theory of well quasi-orderings. One interesting direction for future work is to extend the algorithm so that we solve the more general problem of language inclusion. Another challenge is to extend the algorithm to deal with the case of general (multi-clock) timed automata. In fact, we can modify the notion of zones by adding extra information to keep track of clocks which belong to the same state inside a configuration. In such a case, computing successors and checking entailment can be carried out in a similar manner to the methods of this paper. However, the entailment relation will not be a well quasi-ordering, and therefore the

universality algorithm will no more be guaranteed to terminate. This is expected since the problem is undecidable for multi-clock timed automata. Despite this, using zones may make the algorithm terminate sufficiently often so that it becomes practically interesting even for the general case.

References

1. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
2. Alur, R., Madhusudan, P.: Decision problems for timed automata: A survey. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems. LNCS, vol. 3185, Springer, Heidelberg (2004)
3. Alur, R., Fix, L., Henzinger, T.: Event-clock automata: A determinizable class of timed automata. Theoretical Computer Science 211, 253–273 (1999)
4. Alur, R., Torre, S.L., Madhusudan, P.: Perturbed timed automata. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, Springer, Heidelberg (2005)
5. Alur, R., Feder, T., Henzinger, T.: The benefits of relaxing punctually. Journal of the ACM 43, 116–146 (1996)
6. Henzinger, T., Manna, Z., Pnueli, A.: What good are digital clocks? In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, Springer, Heidelberg (1992)
7. Ouaknine, J., Worrell, J.: Universality and language inclusion for open and closed timed automata. In: Maler, O., Pnueli, A. (eds.) HSCC 2003. LNCS, vol. 2623, Springer, Heidelberg (2003)
8. Ouaknine, J., Worrell, J.: On the language inclusion problem for timed automata: Closing a decidability gap. In: Proc. LICS 2004, 20th IEEE Int. Symp. on Logic in Computer Science, IEEE Computer Society Press, Los Alamitos (2004)
9. Abdulla, P.A., Deneux, J., Ouaknine, J., Worrell, J.: Decidability and complexity results for timed automata via channel machines. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, Springer, Heidelberg (2005)
10. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Information and Computation 127(2), 91–101 (1996)
11. Abdulla, P.A., Nylén, A.: Timed Petri nets and BQOs. In: Colom, J.-M., Koutny, M. (eds.) ICATPN 2001. LNCS, vol. 2075, pp. 53–70. Springer, Heidelberg (2001)
12. Yovine, S.: Kronos: A verification tool for real-time systems. Journal of Software Tools for Technology Transfer 1(1-2) (1997)
13. Larsen, K., Pettersson, P., Yi, W.: Uppaal in a nutshell. Software Tools for Technology Transfer 1(1-2) (1997)
14. Abdulla, P.A., Nylén, A.: Better is better than well: On efficient verification of infinite-state systems. In: Proc. LICS 2000, 16th IEEE Int. Symp. on Logic in Computer Science, pp. 132–140. IEEE Computer Society Press, Los Alamitos (2000)
15. Bengtsson, J., Larsson, F.: Uppaal a tool for automatic verification of real-time systems. Technical Report 96/97, DoCS, Uppsala University (1996)
16. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.K.: Algorithmic analysis of programs with well quasi-ordered domains. Information and Computation 160, 109–127 (2000)

17. Marcone, A.: Foundations of bqo theory. Transactions of the American Mathematical Society 345(2) (1994)
18. Abdulla, P.A., Jonsson, B.: Verifying networks of timed processes. In: Steffen, B. (ed.) ETAPS 1998 and TACAS 1998. LNCS, vol. 1384, pp. 298–312. Springer, Heidelberg (1998)
19. Møller, J., Lichtenberg, J.: Difference decision diagrams. Master's thesis, Department of Information Technology, Technical University of Denmark, Building 344, DK-2800 Lyngby, Denmark (1998)