March 2005

Perturbed Timed Automata

Rajeev Alur
University of Pennsylvania, alur@cis.upenn.edu

Salvatore La Torre
Università degli Studi di Salerno

P. Madhusudan
University of Illinois

Recommended Citation

Rajeev Alur, Salvatore La Torre, and P. Madhusudan, "Perturbed Timed Automata", *Lecture Notes in Computer Science: Hybrid Systems: Computation and Control* 3414, 70-85. March 2005. http://dx.doi.org/10.1007/978-3-540-31954-2_5

From the 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005.

Perturbed Timed Automata

Abstract
We consider timed automata whose clocks are imperfect. For a given perturbation error $0 < \varepsilon < 1$, the *perturbed language* of a timed automaton is obtained by letting its clocks change at a rate within the interval $[1 - \varepsilon, 1 + \varepsilon]$. We show that the perturbed language of a timed automaton with a single clock can be captured by a *deterministic* timed automaton. This leads to a decision procedure for the language inclusion problem for systems modeled as products of 1-clock automata with imperfect clocks. We also prove that determinization and decidability of language inclusion are not possible for multi-clock automata, even with perturbation.

Comments
From the 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005.

Perturbed Timed Automata*

Rajeev Alur[1], Salvatore La Torre[2], and P. Madhusudan[3]

[1] University of Pennsylvania
[2] Università degli Studi di Salerno
[3] University of Illinois at Urbana-Champaign

Abstract. We consider timed automata whose clocks are imperfect. For a given perturbation error $0 < \varepsilon < 1$, the *perturbed language* of a timed automaton is obtained by letting its clocks change at a rate within the interval $[1 - \varepsilon, 1 + \varepsilon]$. We show that the perturbed language of a timed automaton with a single clock can be captured by a *deterministic* timed automaton. This leads to a decision procedure for the language inclusion problem for systems modeled as products of 1-clock automata with imperfect clocks. We also prove that determinization and decidability of language inclusion are not possible for multi-clock automata, even with perturbation.

1 Introduction

Traditional automata do not admit an explicit modeling of time and consequently *timed automata* [1] were introduced as a formal notation to model the behavior of real-time systems. Timed automata are finite automata extended with real-valued variables called *clocks*, whose vertices and edges are annotated with clock constraints that allow specification of constant bounds on delays among events. Timed automata accept *timed languages* consisting of sequences of events tagged with their occurrence times. Over the years, the formalism has been extensively studied leading to many results establishing connections to circuits and logic, and much progress has been made in developing verification algorithms, heuristics, and tools (see [2] for a recent survey and [3–5] for sample tools). The class of timed regular languages —languages definable by timed automata— is closed under union, intersection and projection, but not under complementation, and while language emptiness can be decided by symbolic algorithms manipulating clock constraints, decision problems such as universality and language inclusion are undecidable for timed automata [1].

The undecidability of language inclusion and nonclosure under complementation has motivated many researchers to search for ways to limit the expressiveness of timed automata (see for example [1, 6–12]). A canonical example of a timed regular language whose complement is not timed regular, is the language

* This research was partially supported by the US National Science Foundation under grants ITR/SY0121431 and CCR0410662. The second author was also supported by the MIUR grant ex-60% 2003 Università degli Studi di Salerno.

L^1 of timed words containing *some* two symbols separated *exactly* by 1 time unit. In fact, a single clock suffices to express L^1. Typical proofs of undecidability of language inclusion crucially use the language L^1. One way to avoid L^1 is to require that the automaton be *deterministic*: since there can be unboundedly many symbols in an interval of 1 time unit, nondeterminism is *necessary* to accept L^1. The class of deterministic timed automata is closed under union, intersection, and complementation, and problems such as universality and inclusion are decidable for deterministic timed automata [1]. An alternative way to rule out L^1 is inspired by the observation that L^1 relies on the (infinite) precision of the timing constraints. In *robust timed automata* fuzziness is introduced in the language of an automaton semantically using a *metric* over the timed words, and considering a word to be accepted/rejected only if a *dense* subset around the word is accepted/rejected [13]. Unfortunately, language inclusion remains undecidable under the robust semantics also, and robust languages are not closed under complementation [14].

In this paper, we propose and study an alternative way of introducing imprecision in timed automata by introducing *errors in the rates of clocks*. Given a timed automaton A and a rational constant $0 \leq \varepsilon < 1$, let $L_\varepsilon(A)$ be the language of the automaton in the *perturbed* semantics, where each clock increases at a rate within the interval $[1-\varepsilon, 1+\varepsilon]$. If we add a perturbation ε to the standard timed automaton accepting L^1, then the resulting language consists of timed words with some two symbols separated by a distance d such that $1-\varepsilon \leq d \leq 1+\varepsilon$. Perturbed timed automata can be seen to be special kinds of (initialized) *rectangular automata* [15]. It follows that a perturbed timed automaton can be translated to a timed automaton preserving the timed language, and emptiness of perturbed languages is decidable.

Our main result is that if A has one clock, then the language $L_\varepsilon(A)$, $\varepsilon > 0$, can be accepted by a *deterministic* timed automaton. Intuitively, when the clock has a drift, then instead of guessing the event on which the clock gets reset, it suffices to remember the *first* and the *last* possible times when the reset may occur in every interval of length ε. More precisely, given a 1-clock automaton A with m locations and c as the largest (integer) constant in its clock constraints, and an error $\varepsilon = 1/n$, we show how to construct a deterministic timed automaton B with $O(cmn)$ clocks that accepts the language $L_\varepsilon(A)$. We also prove the construction to be essentially tight via lower bounds on the number of clocks in any equivalent deterministic automaton. This construction, however, does not generalize when A has multiple clocks: we show that for every $\varepsilon > 0$, there exists a timed automaton A with two clocks such that $L_\varepsilon(A)$ is not definable using deterministic timed automata.

Our result leads to a decision procedure for checking inclusion for systems expressed as products of 1-clock automata with perturbation. That is, consider a system A expressed as a product of 1-clock components A_i, and a system B expressed as a product of 1-clock components B_j. Then, given a perturbation error $\varepsilon > 0$, we can test whether the language of the product of $L(A_i)$ is included in the language of the product of $L_\varepsilon(B_j)$, using our translation from 1-clock

perturbed automata to deterministic ones. This procedure requires space that is linearly proportional to $1/\varepsilon$, linearly proportional to the maximum constant c mentioned in the component automata, and polynomial in the size of the automata.

Systems expressed as products of 1-clock nondeterministic timed automata are common. For example, an asynchronous circuit with timing assumptions can be expressed as a product of 1-clock automata modeling individual gates, where the clock measures the time elapsed since the switch to the excited state, and nondeterminism is used to model the unpredictable effect of an input in the excited state (see for example [16–19]). As we explain in the paper, the results on perturbed timed automata can be used for checking inclusion $L(I) \subseteq L(S)$, where I and S are asynchronous circuits with I being a refinement of S, and where they are modeled using products of 1-clock automata. For establishing decidability of this problem, it is crucial that we take product *after* perturbing the components, rather than perturbing the standard product that allows precise synchronization.

Related work. There have been many attempts to introduce errors in timed automata. As mentioned earlier, robust timed automata have been introduced and studied by changing the notion of acceptance using a metric over timed words that allows perturbation of occurrence times of events [13]. The impact of introducing drifts in clocks on reachability is studied by Puri in [20]: a location of a timed automaton A is defined to be *limit-reachable* if, for every $\varepsilon \geq 0$, it is reachable if we let the clocks change at a rate within the interval $[1-\varepsilon, 1+\varepsilon]$, and the paper shows that while limit reachability is different from standard reachability, it can be decided by modifying the search in the region graph. Instead of perturbing the clock rates, if we perturb the guards, and ask if a location is reachable for every perturbation ε of the guards, then the problem is solvable by similar techniques [21]. The benefits of disallowing precise timing constraints have been observed in other contexts also. For example, the model checking problem for real-time linear temporal logics with modalities bounded by intervals becomes decidable if the intervals are required to be non-singular [22], and the requirement for decidability of language emptiness of rectangular automata that all clocks be initialized, can be relaxed if the guards are perturbed [23].

Among the numerous results pertaining to language inclusion for timed automata, the most relevant result for this paper is that checking whether the language of a timed automaton A is contained in that of B is decidable if B has a single clock [12]. This result, in conjunction with translation from initialized rectangular automata to timed automata, however, does not imply decidability of language inclusion problem for single-clock perturbed automata, since the translation doubles the number of clocks. Furthermore, the algorithm in [12] has high complexity and some recent work shows that it must require space that is not even primitive-recursive in the input [24]. Our results hence show that introducing perturbation leads to a sharp drop in complexity for the decision procedures.

2 Perturbed Timed Automata

Let C be a finite set of clocks. The set of clock constraints $\Phi(C)$ is the smallest set that contains:

- $x \leq y + c$, $x \geq y + c$, $x = y + c$, $x \leq c$, $x \geq c$ and $x = c$ for every $x, y \in C$ and rational number c; we call such constraints *atomic* clock constraints;
- $\neg\delta$ and $\delta_1 \wedge \delta_2$ where $\delta, \delta_1, \delta_2 \in \Phi(C)$.

A *clock interpretation* is a mapping $\nu : C \longrightarrow \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of nonnegative real numbers. If ν is a clock interpretation and d is a real number, let $(\nu + d)$ denote the clock interpretation that maps each clock x to $\nu(x) + d$. If $\lambda \subseteq C$, let $[\lambda \rightarrow 0](\nu)$ be the clock interpretation that maps each clock $x \in \lambda$ to 0 and maps each clock $x \notin \lambda$ to $\nu(x)$.

A *timed automaton* A is a tuple $(\Sigma, Q, Q_0, C, \Delta, F)$ where:

- Σ is a finite set of symbols (alphabet);
- Q is a finite set of locations;
- $Q_0 \subseteq Q$ is a set of initial locations;
- C is a finite set of clock variables;
- Δ is a finite subset of $Q \times \Sigma \times \Phi(C) \times 2^C \times Q$ (edges);
- $F \subseteq Q$ is a set of final locations.

A timed automaton is *deterministic* if $|Q_0| = 1$ and for each pair of distinct edges $(q, \sigma, \delta_1, \lambda_1, q_1), (q, \sigma, \delta_2, \lambda_2, q_2) \in \Delta$, $\delta_1 \wedge \delta_2$ is not satisfiable.

A *state* of a timed automaton A is a pair (q, ν) where $q \in Q$ and ν is a clock interpretation. An *initial state* is a state (q_0, ν_0) where $q_0 \in Q_0$ and $\nu_0(x) = 0$ for every $x \in C$. A *final state* is a state (q, ν) where $q \in F$. The semantics of a timed automaton is given by a transition system over the set of its states. The transitions of this system are divided into *discrete steps* and *time steps*. A discrete step is of the form $(q, \nu) \xrightarrow{\sigma} (q', \nu')$ where there is an edge $(q, \sigma, \delta, \lambda, q') \in \Delta$ such that ν satisfies δ and $\nu' = [\lambda \leftarrow 0]\nu$. A time step is of the form $(q, \nu) \xrightarrow{d} (q, \nu')$ where $\nu' = \nu + d$, $d \in \mathbb{R}_+$. A *step* is $(q, \nu) \xrightarrow{\sigma, d} (q', \nu')$ where $(q, \nu) \xrightarrow{d} (q, \nu'')$ and $(q, \nu'') \xrightarrow{\sigma} (q', \nu')$, for some clock interpretation ν''.

A *timed word* (σ, τ) over the alphabet Σ is such that $\sigma \in \Sigma^*$, $\tau \in \mathbb{R}_+^*$, $|\sigma| = |\tau|$, and if $\tau = \tau_1 \ldots \tau_k$, then for each $i < k$, $\tau_i \leq \tau_{i+1}$.

Let (σ, τ) be a timed word with $\sigma = \sigma_1 \ldots \sigma_k$ and $\tau = \tau_1 \ldots \tau_k$. A run r of a timed automaton A on (σ, τ) is a sequence $(q_0, \nu_0) \xrightarrow{\sigma_1, \tau_1} (q_1, \nu_1) \xrightarrow{\sigma_2, \tau_2 - \tau_1} \ldots \xrightarrow{\sigma_k, \tau_k - \tau_{k-1}} (q_k, \nu_k)$.

The timed word (σ, τ) is *accepted* by a timed automaton A if there is a run r of A on (σ, τ) starting from an initial state and ending in a final state. The (timed) language accepted by A, denoted $L(A)$, is defined as the set $\{(\sigma, \tau) \mid (\sigma, \tau)$ is accepted by $A\}$.

Nondeterministic timed automata are more powerful than their deterministic counterparts. For example, consider the language L^1 of timed words over the single symbol a such that there are two occurrences of a one unit apart. A timed

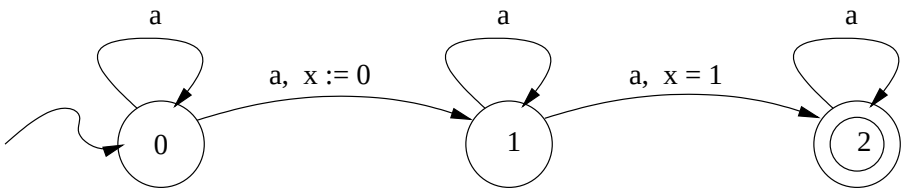

Fig. 1. Timed automaton accepting words with two occurrences of a one unit apart.

automaton accepting L^1 is shown in Figure 1. This automaton nondeterministically guesses an occurrence of a on which it resets the clock x and then checks that there is a following occurrence of a when $x = 1$.

Any deterministic strategy to check a pair of occurrences of a with the above property would need to reset a clock on each occurrence of a. Intuitively, since a clock cannot be reused until time 1 has elapsed and there could be an arbitrary number of a occurrences in a time interval of length 1, in a deterministic automaton we would need to use an unbounded number of clocks, and thus there is no deterministic timed automaton accepting this language (see [2] for a formal proof).

Perturbed semantics for Timed Automata

The clocks of a timed automaton are assumed to be perfect, and all clocks increase at the exact rate 1 with respect to time. We proceed to introduce errors in clock rates to model imprecision.

Let A be a timed automaton $(\Sigma, Q, Q_0, \Delta, C, F)$ and let $0 \leq \varepsilon < 1$ be a rational number. An *ε-perturbed time step* of A is $(q, \nu) \xrightarrow{d}_{\varepsilon} (q, \nu')$ where $\nu(x) + d(1-\varepsilon) \leq \nu'(x) \leq \nu(x) + d(1+\varepsilon)$. An *$\varepsilon$-perturbed step* of A is $(q, \nu) \xrightarrow{\sigma,d}_{\varepsilon} (q', \nu')$ where $(q, \nu) \xrightarrow{d}_{\varepsilon} (q, \nu'')$ and $(q, \nu'') \xrightarrow{\sigma} (q', \nu')$, for some clock interpretation ν''.

An *ε-perturbed run* r of A on a timed word (σ, τ), where $\sigma = \sigma_1 \ldots \sigma_k$ and $\tau = \tau_1 \ldots \tau_k$, is a sequence $(q_0, \nu_0) \xrightarrow{\sigma_1, \tau_1}_{\varepsilon} (q_1, \nu_1) \xrightarrow{\sigma_2, \tau_2 - \tau_1}_{\varepsilon} \ldots \xrightarrow{\sigma_k, \tau_k - \tau_{k-1}}_{\varepsilon} (q_k, \nu_k)$. The ε-perturbed language accepted by A, denoted $L_\varepsilon(A)$, is the language of all the timed words (σ, τ) such that there is an ε-perturbed run r of A on (σ, τ) starting from an initial state and ending in a final state.

As an example of an ε-perturbed language, consider again the timed automaton in Figure 1. For a given ε, the language $L_\varepsilon(A)$ contains all the timed words over the symbol a such that some two a's occur at a distance d, for some $d \in [1-\varepsilon, 1+\varepsilon]$.

Note that according to the definitions, $L_0(A) = L(A)$ for any timed automaton A. Also, note that during a perturbed time step, the drifts in the clocks are independent. Perturbation in the language of a timed automaton can also be expressed by transforming a timed automaton into an initialized rectangular automaton where the rate of change of each clock x is modeled by the differential

inclusion $\dot{x} \in [1-\varepsilon, 1+\varepsilon]$. From the results on rectangular automata, it follows that the timed language of the transformed automaton can be captured by a (nondeterministic) timed automaton [15].

Proposition 1. *For every timed automaton A and a rational constant $0 \leq \varepsilon < 1$, the perturbed language $L_\varepsilon(A)$ is a timed regular language.*

3 Determinization

For the automaton A of Figure 1, while, as observed before, $L(A)$ is not accepted by any deterministic timed automaton, for each $0 < \varepsilon < 1$ it is possible to construct a deterministic timed automaton B that accepts $L_\varepsilon(A)$.

Let us say that two a events, or their occurrence times, are *matching* if they are separated by a distance $d \in [1-\varepsilon, 1+\varepsilon]$. A timed word is in $L_\varepsilon(A)$ if it contains a matching pair. Consider any three events within a time interval of length 2ε occurring respectively at time t_1, t_2 and t_3 with $t_1 < t_2 < t_3$. If an event a occurs at a time $t \in [t_2+1-\varepsilon, t_2+1+\varepsilon]$, then also $t \in [t_1+1-\varepsilon, t_1+1+\varepsilon] \cup [t_3+1-\varepsilon, t_3+1+\varepsilon]$ holds. In other words, if the events at occurrence times t_2 and t are matching, then either the occurrences at t_1 and t are matching, or the occurrences at times t_3 and t are matching. This property is easily shown by observing that since $t_3 - t_1 \leq 2\varepsilon$, we have that $t_3 + 1 - \varepsilon \leq t_1 + 1 + \varepsilon$ and thus the interval $[t_2+1-\varepsilon, t_2+1+\varepsilon]$ is contained in the union of the intervals $[t_1+1-\varepsilon, t_1+1+\varepsilon]$ and $[t_3+1-\varepsilon, t_3+1+\varepsilon]$. This implies that to search for matching pairs, the event at time t_2, and in fact, at any time between t_1 and t_3, is not needed.

This property suggests to split any timed word into intervals such that each interval has length at least 2ε and any two occurrences of a in it are at most 2ε apart from each other. This can be achieved by resetting a clock x^ε every time it exceeds 2ε. A reset of this clock corresponds to the beginning of a new interval. Note that the total length of any $\lceil \frac{1}{2\varepsilon} \rceil + 1$ consecutive intervals is at least $1+\varepsilon$.

Then, we can use separate clocks to remember the time elapsed since the first and the last occurrences of a in each such interval. A clock can be reused once it exceeds $1+\varepsilon$ (recall that the only time constraint in the timed automaton A is $x = 1$). By a simple counting we just need $\lceil \frac{1}{2\varepsilon} \rceil + 1$ pairs of clocks to handle the sampling. Since we need a clock for splitting the timed word into intervals, the deterministic timed automaton B has exactly $2\left(\lceil \frac{1}{2\varepsilon} \rceil + 1\right) + 1$ clocks. The role of the guard $x = 1$ in A is played in B by guards of the form $1-\varepsilon \leq y \leq 1+\varepsilon$, where y is one of the clocks assigned to a 2ε interval.

3.1 Determinization construction for perturbed one-clock automata

In this section, we outline the determinization for the perturbed languages of 1-clock timed automata.

Theorem 1. *Let A be a timed automaton with one clock, c be the largest constant used in A, and Q be the set of A locations. For a rational number $0 <$*

$\varepsilon < 1$, the language $L_\varepsilon(A)$ is accepted by a deterministic timed automaton with $O(\lceil \frac{1}{\varepsilon} \rceil |Q| c)$ clocks and $2^{O(\lceil \frac{1}{\varepsilon} \rceil |Q| c)}$ locations.

Proof. Consider a timed automaton $A = (\Sigma, Q^A, Q_0^A, \{x\}, \Delta^A, F^A)$ with a single clock variable x. For the ease of presentation, we consider here the case when the only constants used in the clock constraints of A are 0 and 1, and the atomic clock constraints using constant 1 are of the form $x \leq 1$, $x < 1$ or $x = 1$. The general case reduces to this case simply constructing an equivalent automaton that keeps track of the integral part of the clock value in the location. (This automaton also needs to reset the clock every time it reaches 1. To trigger the resets, we require that the input words contain a dummy event at each integral time. Note that this does not add to the recognizing power of timed automata [2].)

Fix $n = \lceil \frac{1}{2\varepsilon} \rceil + 1$. Let $Q^A = \{q_1, \ldots, q_m\}$. In the following, we describe the construction of a deterministic automaton $B = (\Sigma, Q^B, \{q_0\}, C^B, \Delta^B, F^B)$ such that $L(B) = L_\varepsilon(A)$.

The set of clocks C^B contains a clock x^ε and clocks y_i^α and z_i^α for all $q_i \in Q^A$ and $\alpha \in \{0, 1, \ldots, n\}$. Clock x^ε is used, as in the example discussed above, to split the input word in intervals such that each interval has length at least 2ε and any two symbols in it are at most 2ε apart from each other.

Let us number modulo $(n + 1)$ these intervals in the order they appear, starting from 0 for the first interval and so on. In the following, we refer to an interval numbered with α also as an *α-interval*.

For a given timed word w as input, consider all the ε-perturbed runs of A on w ending at q_i such that the last reset of x is in the last α-interval. Clock y_i^α is used to store the *maximum* value of x that is reached at the end of the above runs, i.e., this clock is reset in correspondence to the earliest among the last resets of x in the above runs. Similarly, clock z_i^α is used to store the *minimum* value of x that is reached at the end of the above runs, i.e., this clock is reset in correspondence to the latest reset of x in the above runs. Since these two events are at most 2ε time apart from each other, after 1 unit of time has elapsed, any possible value of x that can be reached on an ε-perturbed run of A resetting x at any point in between these events can be reached by resetting x at one of these two extreme points. Thus, sampling these events for each α-interval and for each location q_i suffices to capture all the ε-perturbed runs of A that end in q_i. Also, since the largest constant in the clock constraints of A is 1, the largest value that needs to be compared with y_i^α and z_i^α is $1 + \varepsilon$. Recall that the total length of any n consecutive intervals in the considered splitting is at least $1 + \varepsilon$. Thus, after $n + 1$ intervals these clocks can be reused since they have exceeded the value $1 + \varepsilon$. At this point, in case there are edges of A from q_i on which x is not reset, then when reusing y_i^α (resp. z_i^α), we need to remember in the location of B the fact that the value of the clock is larger than the maximum constant. For this purpose, we just use a bit for each $q_i \in Q^A$.

More precisely, the set of locations of B contains locations of the form $\langle Q, \boldsymbol{a}, \boldsymbol{b}, \alpha \rangle$, where $\alpha \in \{0, 1, \ldots, n\}$ and:

$$Q \subseteq Q^A, \qquad \boldsymbol{a} = \begin{pmatrix} a_1^0 \ \ldots \ a_m^0 \\ \ldots\ldots\ldots \\ \ldots\ldots\ldots \\ a_1^n \ \ldots \ a_m^n \end{pmatrix}, \qquad \boldsymbol{b} = (b_1, \ldots, b_m)$$

Each component b_i is either 0 or 1. Value 1 denotes that from q_i we can take edges as if there is a clock z_i^β whose value is larger than $1 + \varepsilon$. Note that this implies that also the value of y_i^β is larger than $1 + \varepsilon$. Each component a_i^β is 1 if the pair of clocks y_i^β and y_i^β are used, and is 0 otherwise. The set Q is a set of locations of A. In this construction, the component Q is used as in the usual subset construction for determinizing finite automata. In particular, after reading a timed word w, B will reach a location $\langle Q, \boldsymbol{a}, \boldsymbol{b}, \alpha \rangle$ where Q contains all the locations q such that there is an ε-perturbed run of A over w ending at (q, ν) for some clock valuation ν. Component α simply implements a modulo $(n+1)$ counter that stores the number of the current interval and gets incremented whenever clock x^ε is reset.

The construction of B aims at maintaining the following invariant:

P1. For a timed word w, the run of the automaton B on w ends at a state $(\langle Q, \boldsymbol{a}, \boldsymbol{b}, \alpha \rangle, \nu)$ such that
- Q is exactly the set of A locations q_i such that there is an ε-perturbed run of A over w ending at a state (q_i, ν_i);
- $a_i^\beta = 1$ iff there is an ε-perturbed run of A on w ending at a state (q_i, ν_i) such that the last reset of x happened in the last β interval;
- for each $a_i^\beta = 1$, $\nu(y_i^\beta)$ and $\nu(z_i^\beta)$ are the (upper and lower) bounds on the values of x in the A states that can be reached by an ε-perturbed run on w ending at location q_i and such that the last reset of x happened in the last β interval;
- $b_i = 1$ iff there is an ε-perturbed run of A on w ending at a state (q_i, ν_i) such that $\nu_i(x) > 1$.

The initial state q_0 is $\langle Q_0^A, \boldsymbol{a}_0, \boldsymbol{b}_0, 0 \rangle$, where:

$$\boldsymbol{a}_0 = \begin{pmatrix} a_1^0 \ \ldots \ a_m^0 \\ 0 \ \ldots \ 0 \\ \ldots\ldots\ldots \\ 0 \ \ldots \ 0 \end{pmatrix}, \qquad \boldsymbol{b}_0 = (0, \ldots, 0)$$

and $a_i^0 = 1$ if and only if $q_i \in Q_0^A$ (the active clocks are those of the first interval that correspond to the initial locations of A).

The set of final locations F^B is the set of all locations $\langle Q, \boldsymbol{a}, \boldsymbol{b}, \alpha \rangle$ such that $Q \cap F^A \neq \emptyset$.

For describing the edges of B we need to introduce first some notation. We also assume that the guards of A edges are conjunctions of atomic constraints.

This is without loss of generality since the automaton A is nondeterministic and top level disjunction can be modelled with nondeterminism. Let $I_\varepsilon(\delta, y)$ be a mapping that transforms every clock constraint δ involving only x into a clock constraint involving a clock y, as follows:

- if δ is $x \approx c$ with $\approx \in \{<, \leq\}$, then $I_\varepsilon(\delta, y)$ is $y \approx c(1+\varepsilon)$;
- if δ is $x \approx c$ with $\approx \in \{>, \geq\}$, then $I_\varepsilon(\delta, y)$ is $y \approx c(1-\varepsilon)$;
- if $\delta = \delta_1 \wedge \delta_2$, then $I_\varepsilon(\delta, y) = I_\varepsilon(\delta_1, y) \wedge I_\varepsilon(\delta_2, y)$.

For a guard δ and clocks y, z, we denote by $g(\delta, y, z)$ the clock constraint $I_\varepsilon(\delta, y) \vee I_\varepsilon(\delta, z)$. For a B location $s = \langle Q, \boldsymbol{a}, \boldsymbol{b}, \alpha \rangle$, a clock constraint δ over x and a location $q_i \in Q$, let $h(\delta, i, s) = \delta$ if $b_i = 0$ and otherwise, let $h(\delta, i, s)$ be δ with every term $x > 1$ in it replaced by TRUE. For a location $q_i \in Q^A$, we denote by Δ_i the set of all edges contained in Δ^A from q_i. Given an edge e, we denote by δ_e its guard and $d(e) = i$ if the location entered when e is taken is q_i. Given a set X, we denote by $P(X)$ the set of partitions of X into two sets. A two-set partition is denoted by a pair of sets.

Consider a location $s = \langle Q, \boldsymbol{a}, \boldsymbol{b}, \alpha \rangle$. For each $q_i \in Q$ fix a partition (Δ_i', Δ_i'') of Δ_i. For each of such choice of partitions, we insert in Δ_B an edge such that:

- the guard is the conjunction of $x^\varepsilon < 2\varepsilon$ and
  $\bigwedge_{q_i \in Q} \bigwedge_{e \in \Delta_i'} \left(\bigvee_{\beta=0}^{n} (a_i^\beta = 1) \wedge g(h(\delta_e, i, s), y_i^\beta, z_i^\beta) \right) \wedge$
  $\bigwedge_{e \in \Delta_i''} \left(\bigwedge_{\beta=0}^{n} (a_i^\beta = 1) \rightarrow \neg g(h(\delta_e, i, s), y_i^\beta, z_i^\beta) \right)$;
- the destination location is $\langle Q', \boldsymbol{a}', \boldsymbol{b}', \alpha' \rangle$ where:
  - Q' is the set of all q_j such that $j = d(e)$ for some $e \in \Delta_i'$ and $q_i \in Q$;
  - ${a'}_i^\beta = 1$ if and only if either:
    - $\beta = \alpha$ and there is an edge in Δ_i' on which x is reset, or
    - $a_i^\beta = 1$ and there is an edge in Δ_i' on which x is not reset;
      (clocks y_j^β and z_j^β are in use in the new location either if they refer to the current interval and x is reset on a possible edge from the current state, or they inherit the values of previously used clocks that still need to be considered)
  - $\boldsymbol{b}' = \boldsymbol{b}$ (these bits can change only when entering the next interval in the splitting of the input word);
  - $\alpha' = \alpha$;
- clocks are updated according to the following rules:
  - for each edge $e \in \Delta_i'$ from q_i to q_j on which x is reset: if $a_j^\alpha = 0$ then both y_j^α and z_j^α are reset, otherwise only z_j^α is reset; (recall that by y_j^α and z_j^α we wish to capture the time elapsed respectively from the earliest among the last resets and the latest reset of x over all the runs ending at q_j for which the reset happens in the last α-interval. Thus, if the clocks are already in use, we only have to reset the z-clock since the earliest reset is captured when the y-clock starts being used.)
  - let $\bar{\Delta}_j$ be the set of edges e in $\cup_i \Delta_i'$ on which x is not reset and such that $d(e) = j$. Also, for an edge e from a location q_i denote $o(e) = i$. In case there is an edge in $\bar{\Delta}_j$ whose guard contains only atomic constraints

using constant 0 (that is, they are of the form $x \geq 0$ or $x > 0$), then for $\beta \neq \alpha$, clock y_j^β is assigned with the maximum of y_h^β over $h = d(e)$ for $e \in \bar{\Delta}_j$ and clock z_j^β is assigned with the minimum of z_h^β over $h = d(e)$ for $e \in \bar{\Delta}_j$ (we aim to keep the largest possible interval of x-values). In the other cases, we compute for each edge $e \in \Delta_i'$:

* y_e^β as the minimum between $1 + \varepsilon$ and the value of y_h^β for $h = d(e)$, and
* z_e^β as the maximum between $1 - \varepsilon$ and the value of z_h^β for $h = d(e)$, if there is a conjunct of δ_e (the guard of e) of the form $x = 1$, and as the value of z_h^β for $h = d(e)$, otherwise.

The choice of the values y_e^β and z_e^β aims to rule out all runs that cannot be continued with $\bar{\Delta}_j$ edges. Then, for $\beta \neq \alpha$, y_j^β is assigned with the maximum of y_e^β over $\bar{\Delta}_j$ and z_j^β is assigned with the minimum of z_e^β over $\bar{\Delta}_j$. If $x < 1$ is a conjunct of the guards of all the $\bar{\Delta}_j$ edges and the value assigned to a y-clock is $1 + \varepsilon$, we also need to remember that for this clock, its value is actually the supremum of the actual values of x in the represented runs (this can be handled with an additional bit).

With respect to the same partition we also insert in Δ^B edges that differ from the ones described above for the conjunct $x^\varepsilon \geq 2\varepsilon$ instead of $x^\varepsilon < 2\varepsilon$ in their guards, the clock x^ε is reset, and $\alpha' = (\alpha + 1)\,(mod(n+1))$. Moreover, b_j' is set to 1 if there is an i such that there is an edge $e \in \Delta_i'$ from q_i to q_j that does not reset x, and either $a_i^{\alpha'} = 1$ or $b_i = 1$ (i.e., clock $y_i^{\alpha'}$ is active or its value is larger than $1 + \varepsilon$). In fact, in both cases, there is a run of A that reaches q_j with the value of x larger than 1. Also, $a'^{\alpha'}_j$ is set to 1 if and only if there is a edge $e \in \Delta_i'$ from q_i to q_j that resets x.

The automaton B so defined is clearly deterministic (we use disjoint guards on edges from a given location and symbol) but does not respect the definition of a timed automaton. In fact, we use updates (that compute minimum and maximum over clock values) instead of resets. To determine the minimum/maximum over clock values on an edge we can split an edge into several edges each corresponding in turn to a variable being the minimum/maximum. For this purpose, we can just add on each such edge an appropriate conjunct and then rename the clock corresponding to the minimum/maximum with y_j^α. Thus, we are done since clock renaming does not add expressiveness to timed automata (see [25] for example).

It is possible to prove by induction on the number of steps that the above construction preserves the invariant **P1**. Thus, by the definition of F^B, we can conclude that $L(B) = L_\varepsilon(A)$. ■

3.2 Lower Bounds

The determinization was based on "forgetting" events by covering them with extreme events. This idea fails when there are 2 or more clocks in an automaton. To see this consider the automaton A given in Figure 2. For $\varepsilon = \frac{1}{3}$, the

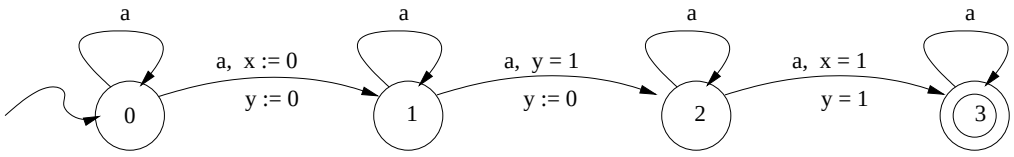

Fig. 2. Automata $\frac{1}{3}$-accepting timed words with two occurrences of a at distance $\frac{3}{2}$.

language $L_\varepsilon(A)$ contains all the timed words over the symbol a such that there is a subsequence aaa where both pairs of events are distance $\frac{3}{4}$ apart. In fact, the only way to fulfill the constraints $x = 1$ and $y = 1$ on the edge from location 2 to location 3 is to let clock y increase at the fastest possible rate (i.e., $\frac{4}{3}$) and clock x increase at the slowest possible rate (i.e., $\frac{2}{3}$). This timed language is basically the same as language L^1 except for the fact that we require that the two occurrences of a are $\frac{3}{2}$ (instead of 1) time apart. Thus, using the same argument as in [2], the complement of the language $L_{\frac{1}{3}}(A)$ cannot be accepted by any timed automaton. Therefore, $L_{\frac{1}{3}}(A)$ cannot be accepted by any deterministic timed automaton as deterministic timed automata are complementable. For any choice of a rational number $\varepsilon \in [0, 1[$, we can generalize the intuition behind the above example and construct a timed automaton that is not complementable. Thus, we have the following:

Proposition 2. *For each perturbation $0 \leq \varepsilon < 1$, there is a timed automaton A with two clocks such that the complement of $L_\varepsilon(A)$ is not accepted by any timed automaton.*

We proceed to show that our construction of determinization for perturbed 1-clock automata is essentially tight. Recall that for a perturbed timed automaton A with locations Q, we built a deterministic timed automaton with $O(\lceil 1/\varepsilon \rceil |Q|)$ clocks. We can show that both these factors are unavoidable:

Theorem 2. *Let $n \in \mathbb{N}$ and let $\varepsilon = 1/n$. Then there exists a 1-clock timed automaton A_n with a constant number of locations such that any deterministic timed automaton B accepting $L_\varepsilon(A_n)$ has at least $n/4$ clocks.*

Proof. Consider the language L^1 consisting of timed words over $\{a\}$ where there are two events a that are one unit apart. It is accepted by the 1-clock nondeterministic timed automaton A shown in Figure 1. Now let $n \in \mathbb{N}$ and $\varepsilon = 1/n$. Let B be a deterministic timed automaton accepting $L_\varepsilon(A)$. Consider an input where there are $n/4$ a events at times $t_1, \ldots t_{n/4}$ where $t_1 = d_1$ and each $t_i = t_{i-1} + 2\varepsilon + d_i$, where each $d_i < \varepsilon$. In order to accept an extension of this word, it is easy to see that an a-event is required in the range $[1, 2]$ in subranges defined by the set of all the values $d_1, \ldots d_{n/4}$. If B uses less than $n/4$ clocks, then there must be some a-event on which a clock was not reset. By making small changes to the values d_i, we can show that B cannot accept the language $L_\varepsilon(A)$. ∎

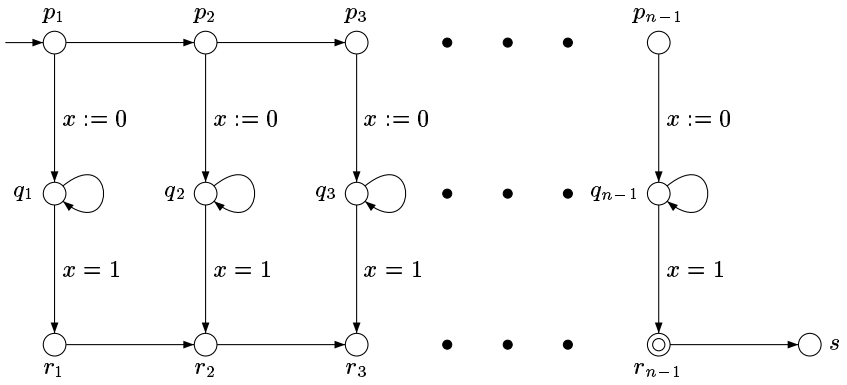

Fig. 3. Automaton used in the lower bound proof of Theorem 3

Theorem 3. *Let $0 < \varepsilon < 1$ be any fixed rational number. For any $n \in \mathbb{N}$, there exists a 1-clock timed automaton A_n with $O(n)$ states such that any deterministic timed automaton B accepting $L_\varepsilon(A_n)$ has at least n clocks.*

Proof. For any n, consider the language over $\Sigma = \{a\}$, consisting all timed words $(a^k, \tau_1, \tau_2, \ldots \tau_k)$ such that there exist $1 \leq i < j \leq k$ with $i + k - j = n$ and $\tau_j - \tau_i = 1$. In other words, there are two events separated by exactly one unit such that the length of the prefix till the first event and the length of the suffix from the latter event add up to n. Figure 3 illustrates a 1-clock timed automaton with $O(n)$ states that guesses these events and accepts the language.

Now consider any deterministic automaton B accepting $L_\varepsilon(A_n)$. Consider a word where n events all before time ε are fed to B. If B had less than n clocks, then there must be some event where a clock was not reset; let this be the i'th event. By suitably extending the word using $n - i$ events after time unit 1 and by timing the first such event after time 1, one can show that B either rejects a word that is in $L_\varepsilon(A_n)$ or accepts a word that is not in $L_\varepsilon(A_n)$. ■

4 Language Inclusion

Let us now consider the inclusion problem for timed automata, which is the problem of deciding whether $L(B) \subseteq L(A)$, for two given timed automata B and A. This question is relevant in the verification context where B can model a timed system and A the safety specification. This problem however turns out to be undecidable; in fact, checking whether $L(A)$ is universal, which is a simpler problem, is itself undecidable [1].

However, if A is a 1-clock timed automaton, then since we can build a deterministic timed automaton A' that accepts the perturbed language of A, it follows that we can decide the language inclusion $L(B) \subseteq L_\varepsilon(A)$ by complementing A', taking its product with B and solving for emptiness. From the results in the previous section, A' has $O(\lceil 1/\varepsilon \rceil |Q| c)$ clocks, if A has locations Q and c is the maximum constant in its guards. Since the emptiness problem for timed automata is in PSPACE, it follows that the inclusion problem can be solved in EXPSPACE. Note that the only exponential factor is in ε and c. For a fixed ε (or if ε was presented in unary) and bounded constants, the inclusion problem is in PSPACE:

Theorem 4. *Given timed automata B and A, where A is a 1-clock automaton, and a perturbation $0 < \varepsilon < 1$, the problem of checking whether $L(B) \subseteq L_\varepsilon(A)$ is decidable in* EXPSPACE. *If ε and the constants in the clock constraints of A are bounded, then the problem is in* PSPACE.

Turning to lower bounds for the above inclusion problem, it is easy to show that the inclusion problem is PSPACE-hard (using a reduction from QBF), and this hardness holds for any fixed ε as well. However, we do not know whether the EXPSPACE upper bound is tight.

The double restriction to 1-clock automata and ε-perturbation is however not necessary to obtain decidability. It turns out that the inclusion problem $L(B) \subseteq L(A)$ is solvable even when A is a 1-clock automaton [12]. However, the decision procedure for this is extremely involved and uses techniques similar to those used in solving questions on (unbounded) Petri nets, and no upper bounds on the complexity are reported. In fact, recent results suggest that the universality problem for 1-clock automata requires non-primitive-recursive space complexity [24]. We note here that the problem is at least EXPSPACE-hard:

Theorem 5. *The universality problem for 1-clock timed automata is* EXPSPACE-*hard.*

Proof. The proof proceeds by a reduction from the membership problem for any EXPSPACE Turing machine. Given an input of length n to such a Turing machine M, we construct a 1-clock timed automaton that accepts the set of all timed words that do *not* correspond to accepting runs of M on that word. Each configuration of M is encoded as a string $c_1 a_1 c_2 a_2 \ldots c_m a_m$ where $a_1 \ldots a_m$ is the contents of the tape cell, m is the space required by M (m is exponential in n) and each c_i is a word of *log* m-bits that encodes the cell number i in binary. A sequence of configurations is then encoded using strings of such sequences. In addition, we require that an encoding of a sequence of configurations be timed correctly, where the distance between a particular bit of c_i in a configuration is encoded exactly one unit from the corresponding bit of c_i in the previous configuration. A timed automaton with $O(n)$ states and 1-clock can easily check if the c_i's in each configuration are encoded correctly, and also check whether the corresponding cells in successive configurations match using the fact that they are exactly one unit of time apart. It follows that this automaton is universal iff M does not accept the input word. ∎

Perturbing 1-clock automata with bounded constants by a fixed ε however results in a simpler determinization construction (non-perturbed 1-clock automata are not determinizable) and a reduction in complexity for the inclusion problem to PSPACE.

The restriction to 1-clock automata is crucial. Recall Proposition 2 which states that there exist automata (in fact with two clocks) such that the complement of its perturbed language is not timed regular. Using the property that using two perturbed clocks one can require two events to be some precise distance apart, we can encode computations of Turing machines to show that:

Theorem 6. *Given timed automata B and A, and a perturbation $\varepsilon > 0$, the problem of deciding whether $L(B) \subseteq L_\varepsilon(A)$ is undecidable.*

4.1 Checking refinement

An application of our results on perturbed timed automata is to check refinement for systems modeled as products of 1-clock automata. Systems such as asynchronous circuits can be modeled using *products of nondeterministic 1-clock automata*: each gate in the circuit is modeled as a timed automaton where the upper and lower bounds on the delay between the excitation of the gate and the triggering of its output is captured using a single clock [16–19]. It is common to model the uncertainty of switching of gates (gates can miss unstable signals, switching of gates can be after varying delays, etc.) using nondeterminism. The asynchronous circuit itself is then a product of 1-clock automata, where the automata synchronize on input-output signals of the respective gates, capturing the design of the circuit.

Consider two systems I and S, each modeled as a product of 1-clock automata, where S is a specification and I is a refinement of S, where some components in S have been implemented using lower level components. We are interested in checking whether all behaviors of I are behaviors of S as well. Let X be the set of events present in the higher level specification S and let I contain events over the set $X \cup Y$, where Y is the new set of events introduced in the implementation.

The problem of checking whether the timed behaviors of I are included in that of S translates into checking if $L(A_I) \subseteq L(A_S)$, where A_I models the behaviors of I and A_S models the behaviors of S in which the new events Y can occur at any time and are ignored. Our results suggest a new way to answer this question. If $A_S = A_1 \| A_2 \| \ldots A_k$, where each A_i is a 1-clock timed automaton, then we can perturb each component A_i of S and then take the product. Such a perturbation is natural in the setting of asynchronous circuits as they anyway model unpredictable perturbation of their signals. We can hence proceed to check whether $L(A_I) \subseteq L_\varepsilon(A_1) \| L_\varepsilon(A_2) \| \ldots L_\varepsilon(A_k)$, which we know is decidable using the results of the previous sections. Notice that in the above expression, we first compute the ε-perturbed languages corresponding to each component and then take the product, which ensures that synchronization is "fudged". This fudging of synchronization is crucial: if we consider $L_\varepsilon(A_1 \| \ldots A_k)$, then since

the automata can synchronize precisely on events, they can accept languages that check whether two events are precisely one unit apart, and the perturbed language of the products of 1-clock automata are not determinizable.

5 Conclusions

Motivated by the gap in the expressiveness in the nondeterministic and deterministic timed automata, and undecidability of the language inclusion problem for nondeterministic timed automata, we initiated the study of timed automata with perturbation in the clock rates. We have proved that one-clock automata are determinizable in presence of perturbation. For systems expressed as products of one-clock automata, this leads to a decidable language inclusion if we perturb individual components. However, if we allow perfect synchronization, and perturb the product, we lose determinization and complementability. The complexity of the inclusion test is exponential in the number of locations as well as the magnitudes of the constants. It remains open whether exponential dependence on the constants, including the perturbation error, can be avoided. There is an alternative way of introducing errors by perturbing the guards of the automaton instead of the clock rates: replace each atomic constraint $x \leq c$ by $x \leq c + \varepsilon$, and $x \geq d$ by $x \geq d - \varepsilon$. The resulting class of perturbed languages has similar properties as the class studied in the paper. Finally, perturbed languages are not closed under projection, and thus, checking language inclusion $L(I) \subseteq L(S)$, when the specification S has internal events not mentioned in the implementation I, is not possible by our techniques even when S is a product of perturbed one-clock components. Thus, checking equivalence of timed circuits composed of components with imperfect clocks, in terms of timed languages over inputs and outputs, remains an interesting open problem.

Acknowledgments We thank Radha Jagadeesan for helpful discussions.

References

1. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235.
2. Alur, R., Madhusudan, P.: Decision problems for timed automata: a survey. In: Formal Methods for the Design of Real-Time Systems. LNCS 3185, Springer (2004) 1–24.
3. Larsen, K., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Springer International Journal of Software Tools for Technology Transfer **1** (1997).
4. Daws, C., Olivero, A., Tripakis, S., Yovine, S.: The tool KRONOS. In: Hybrid Systems III: Verification and Control. LNCS 1066, Springer-Verlag (1996) 208–219.
5. Wang, F.: Efficient data structures for fully symbolic verification of real-time software systems. In: TACAS '00: Sixth Intl Conf on Tools and Algorithms for the Construction and Analysis of Software. LNCS 1785 (2000) 157–171.

6. Henzinger, T., Manna, Z., Pnueli, A.: What good are digital clocks? In: ICALP 92: Automata, Languages, and Programming. LNCS 623. Springer-Verlag (1992) 545–558.
7. Alur, R., Fix, L., Henzinger, T.: Event-clock automata: a determinizable class of timed automata. Theoretical Computer Science **211** (1999) 253–273 A preliminary version appears in *Proc. CAV'94*, LNCS 818, pp. 1–13.
8. Alur, R., Courcoubetis, C., Henzinger, T.: The observational power of clocks. In: CONCUR '94: Fifth International Conference on Concurrency Theory. LNCS 836. Springer-Verlag (1994) 162–177.
9. Alur, R., Henzinger, T.: Back to the future: Towards a theory of timed regular languages. In: Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science. (1992) 177–186.
10. Henzinger, T., Raskin, J., Schobbens, P.: The regular real-time languages. In: ICALP'98: Automata, Languages, and Programming. LNCS 1443. Springer (1998) 580–593.
11. Ouaknine, J., Worrell, J.: Revisiting digitization, robustness, and decidability for timed automata. In: Proc. of the 18th IEEE Symp. on Logic in Comp. Sc. (2003).
12. Ouaknine, J., Worrell, J.: On the language inclusion problem for timed automata: Closing a decidability gap. In: Proceedings of the 19th IEEE Symposium on Logic in Computer Science. (2004).
13. Gupta, V., Henzinger, T., Jagadeesan, R.: Robust timed automata. In: Hybrid and Real Time Systems: International Workshop (HART'97). LNCS 1201, Springer (1997) 48–62.
14. Henzinger, T., Raskin, J.: Robust undecidability of timed and hybrid systems. In: Hybrid Systems: Computation and Control, Third International Workshop. LNCS 1790 (2000) 145–159.
15. Henzinger, T., Kopke, P., Puri, A., Varaiya, P.: What's decidable about hybrid automata. Journal of Computer and System Sciences **57** (1998) 94–124.
16. Brzozowski, J., Seger, C.: Advances in asynchronous circuit theory, Part II: Bounded inertial delay models, MOS circuit design techniques. In: Bulletin of the European Assoc. for Theoretical Comp. Sc. Volume 43. (1991) 199–263.
17. Rokicki, T.: Representing and modeling digital circuits. PhD thesis, Stanford University (1993).
18. Maler, O., Pnueli, A.: Timing analysis of asynchronous circuits using timed automata. In: Proc. of CHARME'95. LNCS 987, Springer (1995) 189–205.
19. Tasiran, S., Brayton, R.: STARI: a case study in compositional and hierarchical timing verification. In: Proceedings of the Ninth International Conference on Computer Aided Verification. LNCS 1254, Springer-Verlag (1997) 191–201.
20. Puri, A.: Dynamical properties of timed automata. In: Proceedings of the 5th International Symposium on Formal Techniques in Real Time and Fault Tolerant Systems. LNCS 1486 (1998) 210–227.
21. De Wulf, M., Doyen, L., Markey, N., Raskin, J.: Robustness and implementability of timed automata. In: Proc. FORMATS. (2004).
22. Alur, R., Feder, T., Henzinger, T.: The benefits of relaxing punctuality. Journal of the ACM **43** (1996) 116–146.
23. Agrawal, M., Thiagarajan, P.S.: Lazy rectangular hybrid automata. In: Hybrid Systems: Computation and Control, Proc. of 7th Intl. Workshop. LNCS 2993, Springer (2004) 1–15.
24. Ouaknine, J.: Personal communication. (2004).
25. Bouyer, P.: Forward analysis of updatable timed automata. Formal Methods in System Design **24** (2004) 281–320.