Efficient Emptiness Check for Timed Büchi Automata

F. Herbreteau, B. Srivathsan, and I. Walukiewicz

LaBRI (Université de Bordeaux -CNRS)

Abstract. The Büchi non-emptiness problem for timed automata concerns deciding if a given automaton has an infinite non-Zeno run satisfying the Büchi accepting condition. The standard solution to this problem involves adding an auxiliary clock to take care of the non-Zenoness. In this paper, we show that this simple transformation may sometimes result in an exponential blowup. We propose a method avoiding this blowup.

1 Introduction

Timed automata [1] are widely used to model real-time systems. They are obtained from finite automata by adding clocks that can be reset and whose values can be compared with constants. The crucial property of timed automata is that their emptiness is decidable. This model has been implemented in verification tools like Uppaal [3] or Kronos [7], and used in industrial case studies [12,4,13].

While most tools concentrate on the reachability problem, the questions concerning infinite executions of timed automata are also of interest. In the case of infinite executions one has to eliminate so called Zeno runs. These are executions that contain infinitely many steps taken in a finite time interval. For obvious reasons such executions are considered unrealistic. In this paper we study the problem of deciding if a given timed automaton has a non-Zeno run passing through accepting states infinitely often. We call this problem *Büchi non-emptiness* problem.

This basic problem has been of course studied already in the paper introducing timed automata. It has been shown that using so called region abstraction the problem can be reduced to the problem of finding a path in a finite region graph satisfying some particular conditions. The main difference between the cases of finite and infinite executions is that in the latter one needs to decide if the path that has been found corresponds to a non Zeno run of the automaton.

Subsequent research has shown that the region abstraction is very inefficient for reachability problems. Another method using zones instead of regions has been proposed. It is used at present in all timed-verification tools. While simple at the first sight, the zone abstraction was delicate to get right. This is mainly because the basic properties of regions do not transfer to zones. The zone abstraction also works for infinite executions, but unlike for regions, it is impossible to decide if a path in a zone graph corresponds to a non-Zeno run of the automaton.

There exists a simple solution to the problem of Zeno runs that amounts to transforming automata in such way that every run passing through an accepting state infinitely often is non-Zeno. An automaton with such a property is called *strongly non-Zeno*. The transformation is easy to describe and requires addition of one new clock. This paper is motivated by our experiments with an implementation of this construction. We have observed that this apparently simple transformation can give a big overhead in the size of a zone graph.

In this paper we closely examine the transformation to strongly non-Zeno automata [17], and show that it can inflict a blowup of the zone graph; and this blowup could even be exponential in the number of clocks. To substantiate, we exhibit an example of an automaton having a zone graph of polynomial size, whose transformed version has a zone graph of exponential size. We propose another solution to avoid this phenomenon. Instead of modifying the automaton, we modify the zone graph. We show that this modification allows us to detect if a path can be instantiated to a non-Zeno run. Moreover the size of the modified graph is $|ZG(\mathcal{A})|.|X|$, where $|ZG(\mathcal{A})|$ is the size of the zone graph and $|X|$ is the number of clocks.

In the second part of the paper we propose an algorithm for testing the existence of accepting non-Zeno runs in timed automata. The problem we face highly resembles the emptiness testing of finite automata with generalized Büchi conditions. Since the most efficient solutions for the latter problem are based on the Tarjan's algorithm, we take the same way here. We present an algorithm whose running time is bounded by $|ZG(\mathcal{A})|.|X|^2$. We also report on the experiments performed with a preliminary implementation of this algorithm.

Related work The zone approach has been introduced in Petri net context [5], and then adapted to the framework of timed automata [9]. The advantage of zones over regions is that they do not require to consider every possible unit time interval separately. The delicate point about zones was to find a right approximation operator. Indeed while regions are both pre- and post-stable, zones are not pre-stable, and some care is needed to guarantee post-stability. Post-stability is enough for correctness of the reachability algorithm, and for testing if a path in the zone graph can be instantiated to a run of the automaton. It is however not possible to determine if a path can be instantiated to a non-Zeno run. The solution involving adding one clock has been discussed in [15,17,2]. Recently, Tripakis [16] has shown a way to extract an accepting run from a zone graph of the automaton. Combined with the construction of adding one clock this gives a solution to our problem. A different approach has been considered in [11] where syntactic conditions are proposed for a timed automaton to be free from Zeno runs. Notice that for obvious complexity reasons, any such condition must be either not complete, or of the same algorithmic complexity as the emptiness test itself.

Organization of the paper In the next section we formalize our problem, and discuss region and zone abstractions. As an intermediate step we give a short proof of the above mentioned result from [16]. Section 3 explains the problems with the transformation to strongly non-Zeno automata, and describes our alternative method. The following section is devoted to a description of the algorithm.

2 The Emptiness Problem for Timed Büchi Automata

2.1 Timed Büchi Automata

Let X be a set of clocks, i.e., variables that range over $\mathbb{R}_{\geq 0}$, the set of non-negative real numbers. *Clock constraints* are conjunctions of comparisons of variables with integer constants, e.g. $(x \leq 3 \wedge y > 0)$. Let $\Phi(X)$ denote the set of clock constraints over clock variables X.

A *clock valuation* over X is a function $\nu : X \to \mathbb{R}_{\geq 0}$. We denote $\mathbb{R}_{\geq 0}^X$ for the set of clock valuations over X, and $\mathbf{0} : X \to \{0\}$ for the valuation that associates 0 to every clock in X. We write $\nu \models \phi$ when ν satisfies ϕ, i.e. when every constraint in ϕ holds after replacing every x by $\nu(x)$.

For a valuation ν and $\delta \in \mathbb{R}_{\geq 0}$, let $(\nu + \delta)$ be the valuation such that $(\nu + \delta)(x) = \nu(x) + \delta$ for all $x \in X$. For a set $R \subseteq X$, let $[R]\nu$ be the valuation such that $([R]\nu)(x) = 0$ if $x \in R$ and $([R]\nu)(x) = \nu(x)$ otherwise.

A *Timed Büchi Automaton (TBA)* is a tuple $\mathcal{A} = (Q, q_0, X, T, Acc)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, X is a finite set of clocks, $Acc \subseteq Q$ is a set of accepting states, and $T \subseteq Q \times \Phi(X) \times 2^X \times Q$ is a finite set of transitions (q, g, R, q') where g is a *guard*, and R is a *reset* of the transition.

A *configuration* of $\mathcal{A}$ is a pair $(q, \nu) \in Q \times \mathbb{R}_{\geq 0}^X$; with $(q_0, \mathbf{0})$ being the *initial configuration*. A *discrete transition* between configurations $(q, \nu) \xrightarrow{t} (q', \nu')$ for $t = (q, g, R, q')$ is defined when $\nu \vDash g$ and $\nu' = [R]\nu$. We also have *delay transitions* between configurations: $(q, \nu) \xrightarrow{\delta} (q, \nu + \delta)$ for every q, ν and $\delta \in \mathbb{R}_{\geq 0}$. We write $(q, \nu) \xrightarrow{\delta, t} (q', \nu')$ if $(q, \nu) \xrightarrow{\delta} (q, \nu + \delta) \xrightarrow{t} (q', \nu')$.

A *run* of $\mathcal{A}$ is a finite or infinite sequence of configurations connected by $\xrightarrow{\delta, t}$ transitions, starting from the initial state q_0 and the initial valuation $\nu_0 = \mathbf{0}$:

$$(q_0, \nu_0) \xrightarrow{\delta_0, t_0} (q_1, \nu_1) \xrightarrow{\delta_1, t_1} \cdots$$

A run σ *satisfies the Büchi condition* if it visits *accepting configurations* infinitely often, that is configurations with a state from Acc. The *duration* of the run is the accumulated delay: $\sum_{i \geq 0} \delta_i$. A run σ is *Zeno* if its duration is bounded.

Definition 1. *The* Büchi non-emptiness problem *is to decide if $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition.*

The class of TBA we consider is usually known as diagonal-free TBA since clock comparisons like $x - y \leq 1$ are disallowed. Since we are interested in the Büchi non-emptiness problem, we can consider automata without an input alphabet and without invariants since they can be simulated by guards.

The Büchi non-emptiness problem is known to be PSPACE-complete [1].

2.2 Regions and region graphs

A simple decision procedure for the Büchi non-emptiness problem builds from $\mathcal{A}$ a graph called the *region graph* and tests if there is a path in this graph satisfying certain conditions. We will define two types of regions.

Fix a constant M and a finite set of clocks X. Two valuations $\nu, \nu' \in \mathbb{R}_{\geq 0}^X$ are *region equivalent* w.r.t. M, denoted $\nu \sim_M \nu'$ iff for every $x, y \in X$:

1. $\nu(x) > M$ iff $\nu'(x) > M$;
2. if $\nu(x) \leq M$, then $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$;
3. if $\nu(x) \leq M$, then $\{\nu(x)\} = 0$ iff $\{\nu'(x)\} = 0$;
4. if $\nu(x) \leq M$ and $\nu(y) \leq M$ then $\{\nu(x)\} \leq \{\nu(y)\}$ iff $\{\nu'(x)\} \leq \{\nu'(y)\}$.

The first three conditions ensure that the two valuations satisfy the same guards. The last one enforces that for every $\delta \in \mathbb{R}_{\geq 0}$ there is $\delta' \in \mathbb{R}_{\geq 0}$, such that valuations $\nu + \delta$ and $\nu' + \delta'$ satisfy the same guards.

We will also define *diagonal region equivalence (d-region equivalence* for short) that strengthens the last condition to

4^d**.** for every integer $c \in (-M, M)$: $\nu(x) - \nu(y) \leq c$ iff $\nu'(x) - \nu'(y) \leq c$

This region equivalence is denoted by $\sim_M^d$. Observe that it is finer than $\sim_M$.

A *region* is an equivalence class of $\sim_M$. We write $[\nu]_{\sim_M}$ for the region of ν, and $\mathcal{R}_M$ for the set of all regions with respect to M. Similarly, for d-region equivalence we write: $[\nu]^d_{\sim_M}$ and $\mathcal{R}^d_M$. If r is a region or a d-region then we will write $r \vDash g$ to mean that every valuation in r satisfies the guard g. Observe that all valuations in a region, or a d-region, satisfy the same guards.

For an automaton $\mathcal{A}$, we define its *region graph*, $RG(\mathcal{A})$, using $\sim_M$ relation, where M is the biggest constant appearing in the guards of its transitions. Nodes of $RG(\mathcal{A})$ are of the form (q, r) for q a state of $\mathcal{A}$ and $r \in \mathcal{R}_M$ a region. There is a transition $(q, r) \xrightarrow{t} (q', r')$ if there are $\nu \in r$, $\delta \in \mathbb{R}_{\geq 0}$ and $\nu' \in r'$ with $(q, \nu) \xrightarrow{\delta, t} (q', \nu')$. Observe that a transition in the region graph is not decorated with a delay. The graph $RG^d(\mathcal{A})$ is defined similarly but using the $\sim_M^d$ relation.

It will be important to understand the properties of pre- and post-stability of regions or d-regions [17]. We state them formally. A transition $(q, r) \xrightarrow{t} (q', r')$ in a region graph or a d-region graph is:

- *Pre-stable* if for every $\nu \in r$ there are $\nu' \in r'$, $\delta \in \mathbb{R}_{\geq 0}$ s.t. $(q, \nu) \xrightarrow{\delta, t} (q', \nu')$.
- *Post-stable* if for every $\nu' \in r'$ there are $\nu \in r$, $\delta \in \mathbb{R}_{\geq 0}$ s.t. $(q, \nu) \xrightarrow{\delta, t} (q', \nu')$.

The following lemma (cf. [6]) explains our interest in $\sim_M^d$ relation.

Lemma 1 (Pre and post-stability). *Transitions in $RG^d(\mathcal{A})$ are pre-stable and post-stable. Transitions in $RG(\mathcal{A})$ are pre-stable but not necessarily post-stable.*

Consider two sequences

$$(q_0, \nu_0) \xrightarrow{\delta_0, t_0} (q_1, \nu_1) \xrightarrow{\delta_1, t_1} \cdots \tag{1}$$

$$(q_0, r_0) \xrightarrow{t_0} (q_1, r_1) \xrightarrow{t_1} \cdots \tag{2}$$

where the first is a run in $\mathcal{A}$, and the second is a path in $RG(\mathcal{A})$ or $RG^d(\mathcal{A})$. We say that the first is an *instance* of the second if $\nu_i \in r_i$ for all $i \geq 0$. Equivalently, we say that the second is an *abstraction* of the first. The following lemma is a direct consequence of the pre-stability property.

Lemma 2. *Every path in $RG(\mathcal{A})$ is an abstraction of a run of $\mathcal{A}$, and conversely, every run of $\mathcal{A}$ is an instance of a path in $RG(\mathcal{A})$. Similarly for $RG^d(\mathcal{A})$.*

This lemma allows us to relate the existence of an accepting run of $\mathcal{A}$ to the existence of paths with special properties in $RG(\mathcal{A})$ or $RG^d(\mathcal{A})$. We say that a path as in (2) *satisfies the Büchi condition* if it has infinitely many occurrences of states from Acc. The path is called *progressive* if for every clock $x \in X$:

- either x is almost always above M: there is n with $r_i \vDash x > M$ for all $i > n$;
- or x is reset infinitely often and strictly positive infinitely often: for every n there are $i, j > n$ such that $r_i \vDash (x = 0)$ and $r_j \vDash (x > 0)$.

Theorem 1 ([1]). *For every TBA $\mathcal{A}$, $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $RG(\mathcal{A})$ has a progressive path satisfying the Büchi condition. Similarly for $RG^d(\mathcal{A})$.*

While this theorem gives an algorithm for solving our problem, it turns out that this method is very impractical. The number of regions $\mathsf{RA}(\mathcal{A})$ is $\mathcal{O}(|X|!2^{|X|}M^{|X|})$ [1] and constructing all of them, or even searching through them on-the-fly, has proved to be very costly.

2.3 Zones and zone graphs

Timed verification tools use zones instead of regions. A zone is a set of valuations defined by a conjunction of two kinds of constraints : comparison of the difference between two clocks with a constant, or comparison of the value of a single clock with a constant. For example $(x - y \geq 1) \wedge (y < 2)$ is a zone. While at first sight it may seem that there are more zones than regions, this is not the case if we count only those that are reachable from the initial valuation.

Since zones are sets of valuations defined by constraints, one can define discrete and delay transitions directly on zones. For $\delta \in \mathbb{R}_{\geq 0}$, we have $(q, Z) \xrightarrow{\delta} (q, Z')$ if Z' is the smallest zone containing the set of all the valuations $\nu + \delta$ with $\nu \in Z$. Similarly, for a discrete transition we put $(q, Z) \xrightarrow{t} (q', Z')$ if Z' is the set of all the valuations ν' such that $(q, \nu) \xrightarrow{t} (q', \nu')$ for some $\nu \in Z$; Z' is a zone in this case. Moreover zones can be represented using Difference Bound Matrices (DBMs), and transitions can be computed efficiently on DBMs [9]. The problem is that the number of reachable zones is not guaranteed to be finite [8].

In order to ensure that the number of reachable zones is finite, one introduces abstraction operators. We mention the three most common ones in the literature. They refer to region graphs, $RG(\mathcal{A})$ or $RG^d(\mathcal{A})$, and use the constant M that is the maximal constant appearing in the guards of $\mathcal{A}$.

- $Closure_M(Z)$: the smallest union of regions containing Z;
- $Closure^d_M(Z)$: similarly but for d-regions;
- $Approx_M(Z)$: the smallest union of d-regions that is convex and contains Z.

The following lemma establishes the links between the three abstraction operators, and is very useful to transpose reachability results from one abstraction to the other.

Lemma 3 ([6]). *For every zone Z: $Z \subseteq Closure^d_M(Z) \subseteq Approx_M(Z) \subseteq Closure_M(Z)$.*

A symbolic zone S is a zone approximated with one of the above abstraction operators. Now, similar to region graphs, we define simulation graphs where after every transition a specific approximation operation is used, that is each node in the simulation graph is of the form (q, S) with S being a symbolic zone . So we have three graphs corresponding to the three approximation operations.

Take an automaton $\mathcal{A}$ and let M be the biggest constant appearing in the guards of its transitions. In the simulation graph $SG(\mathcal{A})$ the nodes are of the form $(q, Closure_M(Z))$ where q is a state of $\mathcal{A}$ and Z is a zone. The initial node is $(q_0, Closure_M(Z_0))$, with q_0 the initial state of $\mathcal{A}$, and Z_0 the zone setting all the variables to 0. The transitions in the graph are $(q, Closure_M(Z)) \xrightarrow{t} (q', Closure_M(Z'))$ where Z' is the set of valuations ν' such that there exist $\nu \in Closure_M(Z)$ and $\delta \in \mathbb{R}_{\geq 0}$ satisfying $(q, \nu) \xrightarrow{\delta,t} (q', \nu')$. Similarly for $SG^d(\mathcal{A})$ and $SG^a(\mathcal{A})$ but replacing $Closure_M$ with operations $Closure^d_M$ and $Approx_M$, respectively. The notions of an abstraction of a run of $\mathcal{A}$, and an instance of a path in a simulation graph are defined in the same way as that of region graphs.

Tools like Kronos or Uppaal use $Approx_M$ abstraction. The two others are less interesting for implementations since the result may not be convex. Nevertheless, they are useful in proofs. The following lemma (cf. [8]) says that transitions in $SG(\mathcal{A})$ are post-stable with respect to regions.

Lemma 4. *Let $(q, S) \xrightarrow{t} (q', S')$ be a transition in $SG(\mathcal{A})$. For every region $r' \subseteq S'$, there is a region $r \subseteq S$ such that $(q, r) \xrightarrow{t} (q', r')$ is a transition in $RG(\mathcal{A})$.*

We get a correspondence between paths in simulation graphs and runs of $\mathcal{A}$.

Theorem 2 ([16]). *Every path in $SG(\mathcal{A})$ is an abstraction of a run of $\mathcal{A}$, and conversely, every run of $\mathcal{A}$ is an instance of a path in $SG(\mathcal{A})$. Similarly for SG^d and SG^a.*

Proof. We first show that a path in $SG(\mathcal{A})$ is an abstraction of a run of $\mathcal{A}$. Take a path $(q_0, S_0) \xrightarrow{t_0} (q_1, S_1) \xrightarrow{t_1} \dots$ in $SG(\mathcal{A})$. Construct a DAG with nodes (i, q_i, r_i) such that r_i is a region in S_i. We put an edge from (i, q_i, r_i) to $(i+1, q_{i+1}, r_{i+1})$ if $(q_i, r_i) \xrightarrow{t_i} (q_{i+1}, r_{i+1})$. By Lemma 4, every node in this DAG has at least one predecessor, and the branching of every node is bounded by the number of regions. Hence, this DAG has an infinite path that is a path in $RG(\mathcal{A})$. By Lemma 2 this path can be instantiated to a run of $\mathcal{A}$.

To conclude the proof one shows that a run of $\mathcal{A}$ can be abstracted to a path in $SG^d(\mathcal{A})$. Then using Lemma 3 this path can be converted to a path in $SG^a(\mathcal{A})$, and later to one in $SG(\mathcal{A})$. We omit the details.

Observe that this theorem does not guarantee that a path we find in a simulation graph has an instantiation that is non-Zeno. It is indeed impossible to

guarantee this unless some additional conditions on paths or automata are imposed.

In the subsequent sections, we are interested only in the simulation graph SG^a. Observe that the symbolic zone obtained by the approximation of a zone using $Approx_M$ is in fact a zone. Hence, we prefer to call it a zone graph and denote it as ZG^a. Every node of ZG^a is of the form (q, Z) where Z is a zone.

3 Finding non Zeno paths

As we have remarked above, in order to use Theorem 2 we need to be sure that a path we get can be instantiated to a non-Zeno run. We discuss the solutions proposed in the literature, and then offer a better one. Thanks to pre-stability of the region graph, the progress criterion on regions has been defined in [1] for selecting runs from $RG(\mathcal{A})$ that have a non-Zeno instantiation (see Section 2.2). Notice that the semantics of TBA in [1] constrains all delays δ_i to be strictly positive, but the progress criterion can be extended to the stronger semantics that is used nowadays (see [17] for instance). However, since zone graphs are not pre-stable, this method cannot be directly extended to zone graphs.

3.1 Adding one clock

A common solution to deal with Zeno runs is to transform an automaton into a *strongly non-Zeno automaton*, i.e. such that all runs satisfying the Büchi condition are guaranteed to be non-Zeno. We present this solution here and discuss why, although simple, it may add an exponential factor in the decision procedure.

The transformation of $\mathcal{A}$ into a strongly non-Zeno automaton $SNZ(\mathcal{A})$ proposed in [17] adds one clock z and duplicates accepting states. One copy of the state is as before but is no longer accepting. The other copy is accepting, but it can be reached only when $z \geq 1$. Moreover when it is reached z is reset to 0. The only transition from this second copy leads to the first copy. (See $\mathcal{V}_k$ and $\mathcal{W}_k$ on Figure 1 for an example.) This construction ensures that at least one unit of time has passed between two visits to an accepting state. A slightly different construction is mentioned in [2]. Of course one can also have other modifications, and it is impossible to treat all the imaginable constructions at once. Our objective here is to show that the constructions proposed in the literature produce a phenomenon that causes proliferation of zones that can sometimes be exponential in the number of clocks. The discussion below will focus on the construction from [17], but the one from [2] suffers from the same problem.

The problem comes from the fact that the constraint $z \geq 1$ may be a source of rapid multiplication of the number of zones in the zone graph of $SNZ(\mathcal{A})$. Consider $\mathcal{V}_k$ and $\mathcal{W}_k$ from Figure 1 for $k = 2$. Starting at the state b^2 of $\mathcal{V}_2$ in the zone $0 \leq y \leq x_1 \leq x_2$, there are two reachable zones with state b^2. Moreover, if we reset x_1 followed by y from the two zones, we reach the same zone $0 \leq y \leq x_1 \leq x_2$. In contrast starting in b^2 of $\mathcal{W}_2 = SNZ(\mathcal{V}_2)$ from

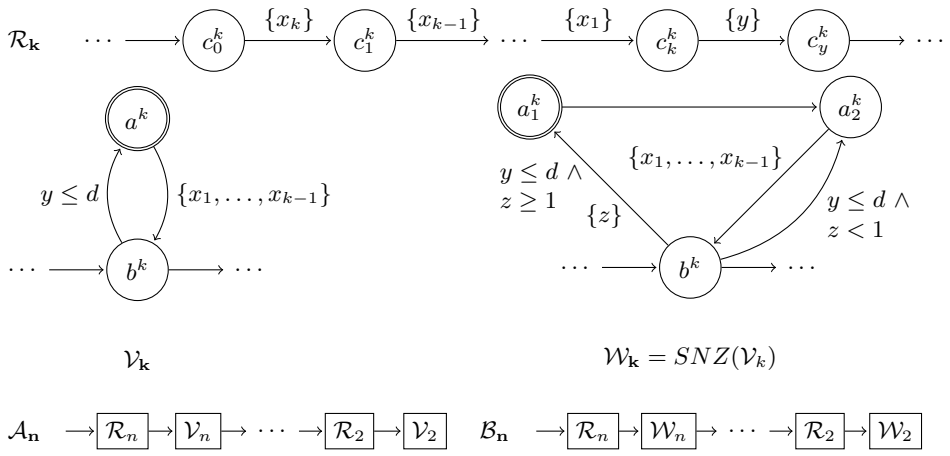

Fig. 1. Gadgets for $\mathcal{A}_n$ and $\mathcal{B}_n = SNZ(\mathcal{A}_n)$.

$0 \le y \le x_1 \le x_2 \le z$ gives at least d zones, and resetting x_1 followed by y still yields d zones.

We now exploit this fact to give an example of a TBA $\mathcal{A}_n$ whose zone graph has a number of zones linear in the number of clocks, but $\mathcal{B}_n = SNZ(\mathcal{A}_n)$ has a zone graph of size exponential in the number of clocks. $\mathcal{A}_n$ is constructed from the automata gadgets $\mathcal{V}_k$ and $\mathcal{R}_k$ as shown in Figure 1. Observe that the role of $\mathcal{R}_k$ is to enforce an order $0 \le y \le x_1 \le \cdots \le x_k$ between clock values. By induction on k one can compute that there are only two zones at locations b^k since $\mathcal{R}_{k+1}$ made the two zones in b^{k+1} collapse into the same zone in b^k. Hence the number of nodes in the zone graph of $\mathcal{A}_n$ is $\mathcal{O}(n)$.

Let us now consider $\mathcal{B}_n$, the strongly non-Zeno automaton obtained from $\mathcal{A}_n$ following [17]. Every gadget $\mathcal{V}_k$ gets transformed to $\mathcal{W}_k$. While exploring $\mathcal{W}_k$, one introduces a distance between the clocks x_{k-1} and x_k. So when leaving it one gets zones with $x_k - x_{k-1} \ge c$, where $c \in \{0, 1, 2, \ldots, d\}$. The distance between x_k and x_{k-1} is preserved by $\mathcal{R}_k$. In consequence, $\mathcal{W}_n$ produces at least $d+1$ zones. For each of these zones $\mathcal{W}_{n-1}$ produces $d+1$ more zones. In the end, the zone graph of $\mathcal{B}_n$ has at least $(d+1)^{n-1}$ zones at the state b^2.

We have thus shown that $\mathcal{A}_n$ has $\mathcal{O}(n)$ zones while $\mathcal{B}_n = SNZ(\mathcal{A}_n)$ has an exponential number of zones even when the constant d is 1. Observe that the construction shows that even with two clocks the number of zones blows exponentially in the binary representation of d. Note that the automaton $\mathcal{A}_n$ does not have a non-Zeno accepting run. Hence, every search algorithm is compelled to explore all the zones of $\mathcal{B}_n$.

3.2 A more efficient solution

Our solution stems from a realization that we only need one non-Zeno run satisfying the Büchi condition and so in a way transforming an automaton to strongly

non-Zeno is an overkill. We propose not to modify the automaton, but to introduce additional information to the zone graph $ZG^a(\mathcal{A})$. The nodes will now be triples (q, Z, Y) where $Y \subseteq X$ is the set of clocks that can potentially be equal to 0. It means in particular that other clock variables, i.e. those from $X - Y$ are assumed to be bigger than 0. We write $(X - Y) > 0$ for the constraint saying that all the variables in $X - Y$ are not 0.

Definition 2. *Let $\mathcal{A}$ be a TBA over a set of clocks X. The* guessing zone graph *$GZG^a(\mathcal{A})$ has nodes of the form (q, Z, Y) where (q, Z) is a node in $ZG^a(\mathcal{A})$ and $Y \subseteq X$. The initial node is (q_0, Z_0, X), with (q_0, Z_0) the initial node of $ZG^a(\mathcal{A})$. There is a transition $(q, Z, Y) \xrightarrow{t} (q', Z', Y \cup R)$ in $GZG^a(\mathcal{A})$ if there is a transition $(q, Z) \xrightarrow{t} (q', Z')$ in $ZG^a(\mathcal{A})$ with $t = (q, g, R, q')$, and there are valuations $\nu \in Z$, $\nu' \in Z'$, and δ such that $\nu + \delta \models (X - Y) > 0$ and $(q, \nu) \xrightarrow{\delta, t} (q, \nu')$. We also introduce a new auxiliary letter τ, and put transitions $(q, Z, Y) \xrightarrow{\tau} (q, Z, Y')$ for $Y' = \emptyset$ or $Y' = Y$.*

Observe that the definition of transitions reflects the intuition about Y we have described above. Indeed, the additional requirement on the transition $(q, Z, Y) \xrightarrow{t} (q', Z', Y \cup R)$ is that it should be realizable when the clocks outside Y are strictly positive; so there should be a valuation satisfying $(X - Y) > 0$ that realizes this transition. As we will see later, this construction entails that from a node $(q, Z, \emptyset)$ every reachable zero-check is preceded by the reset of the variable that is checked, and hence nothing prevents a time elapse in this node. A node of the form $(q, Z, \emptyset)$ is called *clear*. We call a node (q, Z, Y) *accepting* if it contains an accepting state q.

Example. Figure 2 depicts a TBA $\mathcal{A}_1$ along with its zone graph $ZG^a(\mathcal{A}_1)$ and its guessing zone graph $GZG^a(\mathcal{A}_1)$ where τ-loops have been omitted.

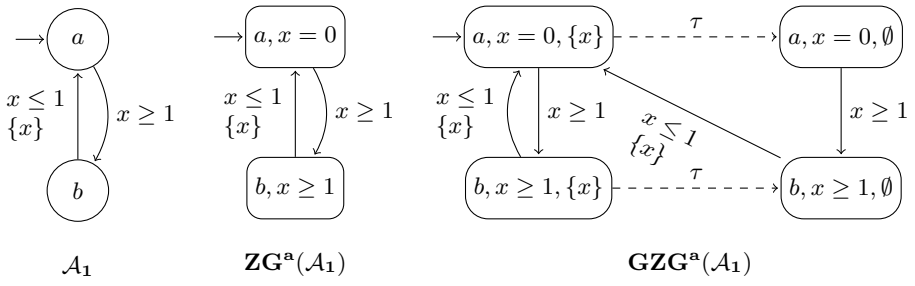

Fig. 2. A TBA $\mathcal{A}_1$ and the guessing zone graph $GZG^a(\mathcal{A}_1)$.

Notice that directly from the definition it follows that a path in $GZG^a(\mathcal{A})$ determines a path in $ZG^a(\mathcal{A})$ obtained by removing τ transitions and the third component from nodes.

A variable x is *bounded* by a transition of $GZG^a(\mathcal{A})$ if the guard of the transition implies $x \leq c$ for some constant c. More precisely: for $(q, Z, Y) \xrightarrow{(q,g,R,q')} (q', Z', Y')$, the guard g implies $(x \leq c)$. A variable is *reset* by the transition if it belongs to the reset set R of the transition.

We say that a path is *blocked* if there is a variable that is bounded infinitely often and reset only finitely often by the transitions on the path. Otherwise the path is called *unblocked*.

Theorem 3. *A TBA $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition iff there exists an unblocked path in $GZG^a(\mathcal{A})$ visiting both an accepting node and a clear node infinitely often.*

The proof of Theorem 3 follows from Lemmas 5 and 6 below. We omit the proof of the first of the two lemmas and concentrate on a more difficult Lemma 6. It is here that the third component of states is used.

At the beginning of the section we had recalled that the progress criterion in [1] characterizes the paths in region graphs that have non-Zeno instantiations. We had mentioned that it cannot be directly extended to zone graphs since their transitions are not pre-stable. Lemma 6 below shows that by slightly complicating the zone graph we can recover a result very similar to Lemma 4.13 in [1].

Lemma 5. *If $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition, then in $GZG^a(\mathcal{A})$ there is an unblocked path visiting both an accepting node and a clear node infinitely often.*

Lemma 6. *Suppose $GZG^a(\mathcal{A})$ has an unblocked path visiting infinitely often both a clear node and an accepting node then $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition.*

Proof. Let σ be a path in $GZG^a(\mathcal{A})$ as required by the assumptions of the lemma (without loss of generality we assume every alternate transition is a τ transition):

$$(q_0, Z_0, Y_0) \xrightarrow{\tau} (q_0, Z_0, Y_0') \xrightarrow{t_0} \cdots (q_i, Z_i, Y_i) \xrightarrow{\tau} (q_i, Z_i, Y_i') \xrightarrow{t_i} \cdots$$

Take a corresponding path in $ZG^a(\mathcal{A})$ and one instance $\rho = (q_0, \nu_0), (q_1, \nu_1) \ldots$ that exists by Theorem 2. If it is non-Zeno then we are done.

Suppose ρ is Zeno. Let X^r be the set of variables reset infinitely often on σ. By assumption on σ, every variable not in X^r is bounded only finitely often. Since ρ is Zeno, there is an index m such that the duration of the suffix of the run starting from (q_m, ν_m) is bounded by $1/2$, and no transition in this suffix bounds a variable outside X^r. Let $n > m$ be such that every variable from X^r is reset between m and n. Observe that $\nu_n(x) < 1/2$ for every $x \in X^r$.

Take positions i, j such that $i, j > n$, $Y_i = Y_j = \emptyset$ and all the variables from X^r are reset between i and j. We look at the part of the run ρ:

$$(q_i, \nu_i) \xrightarrow{\delta_i, t_i} (q_{i+1}, \nu_{i+1}) \xrightarrow{\delta_{i+1}, t_{i+1}} \ldots (q_j, \nu_j)$$

and claim that every sequence of the form

$$(q_i, \nu'_i) \xrightarrow{\delta_i, t_i} (q_{i+1}, \nu'_{i+1}) \xrightarrow{\delta_{i+1}, t_{i+1}} \dots (q_j, \nu'_j)$$

is a part of a run of $\mathcal{A}$ provided there is $\zeta \in \mathbb{R}_{\geq 0}$ such that the following three conditions hold for all $k = i, \dots, j$:

1. $\nu'_k(x) = \nu_k(x) + \zeta + 1/2$ for all $x \notin X^r$,
2. $\nu'_k(x) = \nu_k(x) + 1/2$ if $x \in X^r$ and x has not been reset between i and k.
3. $\nu'_k(x) = \nu_k(x)$ otherwise, i.e., when $x \in X^r$ and x has been reset between i and k.

Before proving this claim, let us explain how to use it to conclude the proof. The claim shows that in (q_i, ν_i) we can pass $1/2$ units of time and then construct a part of the run of $\mathcal{A}$ arriving at (q_j, ν'_j) where $\nu'_j(x) = \nu_j(x)$ for all variables in X^r, and $\nu'_j(x) = \nu_j(x) + 1/2$ for other variables. Now, we can find $l > j$, so that the pair (j, l) has the same properties as (i, j). We can pass $1/2$ units of time in j and repeat the above construction getting a longer run that has passed $1/2$ units of time twice. This way we construct a run that passes $1/2$ units of time infinitely often. By the construction it passes also infinitely often through accepting nodes.

It remains to prove the claim. Take a transition $(q_k, \nu_k) \xrightarrow{\delta_k, t_k} (q_{k+1}, \nu_{k+1})$ and show that $(q_k, \nu'_k) \xrightarrow{\delta_k, t_k} (q_{k+1}, \nu'_{k+1})$ is also a transition allowed by the automaton. Let g and R be the guard of t_k and the reset of t_k, respectively.

First we need to show that $\nu'_k + \delta_k$ satisfies the guard of t_k. For this, we need to check if for every variable $x \in X$ the constraints in g concerning x are satisfied. We have three cases:

- If $x \notin X^r$ then x is not bounded by the transition t_k, that means that in g the constraints on x are of the form $(x > c)$ or $(x \geq c)$. Since $(\nu_k + \delta_k)(x)$ satisfies these constraints so does $(\nu'_k + \delta_k)(x) \geq (\nu_k + \delta_k)(x)$.
- If $x \in X^r$ and it is reset between i and k then $\nu'_k(x) = \nu_k(x)$ so we are done.
- Otherwise, we observe that $x \notin Y_k$. This is because $Y_i = \emptyset$, and then only variables that are reset are added to Y. Since x is not reset between i and k, it cannot be in Y_k. By definition of transitions in $GZG^a(\mathcal{A})$ this means that $g \wedge (x > 0)$ is consistent. We have that $0 \leq (\nu_k + \delta_k)(x) < 1/2$ and $1/2 \leq (\nu'_k + \delta_k)(x) < 1$. So $\nu'_k + \delta_k$ satisfies all the constraints in g concerning x as $\nu_k + \delta_k$ does.

This shows that there is a transition $(q_k, \nu'_k) \xrightarrow{\delta'_k, t_k} (q_{k+1}, \nu')$ for the uniquely determined $\nu' = [R](\nu'_k + \delta_k)$. It is enough to show that $\nu' = \nu'_{k+1}$. For variables not in X^r it is clear as they are not reset. For variables that have been reset between i and k this is also clear as they have the same values in ν'_{k+1} and ν'. For the remaining variables, if a variable is not reset by the transition t_k then condition (2) holds. If it is reset then its value in ν' becomes 0; but so it is in ν'_{k+1}, and so the third condition holds. This proves the claim.

Finally, we provide an explanation as to why the proposed solution does not produce an exponential blowup. At first it may seem that we have gained nothing because when adding arbitrary sets Y we have automatically caused exponential blowup to the zone graph. We claim that this is not the case for the part of $GZG^a(\mathcal{A})$ reachable from the initial node, namely a node with the initial state of $\mathcal{A}$ and the zone putting every clock to 0.

We say that a zone *orders clocks* if for every two clocks x, y, the zone implies that at least one of $x \leq y$, or $y \leq x$ holds.

Lemma 7. *If a node with a zone Z is reachable from the initial node of the zone graph $ZG^a(\mathcal{A})$ then Z orders clocks. The same holds for $GZG^a(\mathcal{A})$.*

Suppose that Z orders clocks. We say that a set of clocks Y *respects the order given by* Z if whenever $y \in Y$ and Z implies $x \leq y$ then $x \in Y$.

Lemma 8. *If a node (q, Z, Y) is reachable from the initial node of the zone graph $GZG^a(\mathcal{A})$ then Y respects the order given by Z.*

Lemma 7 follows since a transition from a zone that orders clocks gives back a zone that orders clocks, and the $Approx_M$ operator approximates it again to a zone that orders clocks. Notice that the initial zone clearly orders clocks. The proof of Lemma 8 proceeds by an induction on the length of a path. The above two lemmas give us the desired bound.

Theorem 4. *Let $|ZG^a(\mathcal{A})|$ be the size of the zone graph, and $|X|$ be the number of clocks in $\mathcal{A}$. The number of reachable nodes of $GZG^a(\mathcal{A})$ is bounded by $|ZG^a(\mathcal{A})|.(|X|+1)$.*

The theorem follows directly from the above two lemmas. Of course, imposing that zones have ordered clocks in the definition of $GZG^a(\mathcal{A})$ we would get the same bound for the entire $GZG^a(\mathcal{A})$.

4 Algorithm

We use Theorem 3 to construct an algorithm to decide if an automaton $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition. This theorem requires to find an unblocked path in $GZG^a(\mathcal{A})$ visiting both an accepting state and a clear state infinitely often. This problem is similar to that of testing for emptiness of automata with generalized Büchi conditions as we need to satisfy two infinitary conditions at the same time. The requirement of a path being unblocked adds additional complexity to the problem. The best algorithms for testing emptiness of automata with generalized Büchi conditions are based on strongly connected components (SCC) [14,10]. So this is the way we take here.

We apply a variant of Tarjan's algorithm for detecting maximal SCCs in $GZG^a(\mathcal{A})$. During the computation of the maximal SCCs, we keep track of whether an accepting node and a clear node have been seen. For the unblocked condition we use two sets of clocks UB_Γ and R_Γ that respectively contain the clocks that are bounded and the clocks that are reset in the SCC Γ. At the end of the exploration of Γ we check if:

1. we have passed through an accepting node and a clear node,
2. there are no blocking clocks: $UB_\Gamma \subseteq R_\Gamma$.

If the two conditions are satisfied then we can conclude saying that $\mathcal{A}$ has an accepting non-Zeno run. Indeed, a path passing infinitely often through all the nodes of Γ would satisfy the conditions of Theorem 3, giving a required run of $\mathcal{A}$. If the first condition does not hold then the same theorem says that Γ does not have a witness for a non-Zeno run of $\mathcal{A}$ satisfying the Büchi condition.

The interesting case is when the first condition holds but not the second. We can then discard from Γ all the edges blocking clocks from $UB_\Gamma - R_\Gamma$, and reexamine it. If Γ without discarded edges is still an SCC then we are done. If not we restart our algorithm on Γ with the discarded edges removed. Observe that we will not do more than $|X|$ restarts, as each time we eliminate at least one clock. If after exploring the entire graph, the algorithm has not found a subgraph satisfying the two conditions then it declares that there is no run of $\mathcal{A}$ with the desired properties. Its correctness is based on Theorem 3.

Recall that by Theorem 4 the size of $GZG^a(\mathcal{A})$ is $|ZG^a(\mathcal{A})| \cdot |X|$. The complexity of the algorithm follows from the complexity of Tarjan's algorithm and the remark about the number of recursive calls being bounded by the number of clocks. We hence obtain the following.

Theorem 5. *The above algorithm is correct and runs in time* $\mathcal{O}(|ZG^a(\mathcal{A})|.|X|^2)$.

5 Conclusions

Büchi non-emptiness problem is one of the standard problems for timed automata. Since the paper introducing the model, it has been widely accepted that addition of one auxiliary clock is an adequate method to deal with the problem of Zeno paths. This technique is also used in the recently proposed zone based algorithm for the problem [16].

In this paper, we have argued that in some cases the auxiliary clock may cause exponential blowup in the size of the zone graph. We have proposed another method that is based on a modification of the zone graph. The resulting graph grows only by a factor that is linear in the number of clocks. In our opinion, the efficiency gains of our method outweigh the fact that it requires some small modifications in the code dealing with zone graph exploration.

It is difficult to estimate how often introducing the auxiliary clock may cause an exponential blowup. The example in Figure 1 suggests that the problem appears when there is a blocked cycle containing an accepting state. A prototype implementation of our algorithm shows that a blowup occurs in the Train-Gate example (e.g. [11]) when checking for bounded response to train access requests. For 2 trains, the zone graph has 988 zones whereas after adding the auxiliary clock it blows to 227482 zones. The guessing zone graph has 3840 zones. To be fair, among the 227482 zones, 146061 are accepting with a self-loop, so in this example any on-the-fly algorithm should work rather quickly. Our prototype visits 1677 zones (in $0.42s$). The example from Figure 1 with $n = 10$ and $d = 1$

has a zone graph with 151 zones and a guessing zone graph with 1563 zones. Its size grows to 36007 when adding the extra clock. Raising d to 15, we obtain 151, 1563 and 135444 zones respectively, which confirms the expected blowup.

It is possible to apply the modification to the zone graph on-the-fly. It could also be restricted only to strongly connected components having "zero checks". This seems to be another source of big potential gains. We are currently working on an on-the-fly optimized algorithm. The first results are very encouraging. Often our prototype implementation solves the emptiness problem at the same cost as reachability when the automaton has no Zeno accepting runs. For instance, the zone graph for Fischer's protocol with 4 processes has 46129 zones and is computed in $14.22s$[1]. To answer the mutual exclusion problem it is necessary to visit the entire zone graph. Our algorithm does it in $15.77s$. Applying the construction from [17] we get the graph with 96913 zones, and it takes $37.10s$ to visit all of them. Hence, even in this example, where all the runs are non-Zeno, adding one clock has a noticeable impact.

References

1. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. R. Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *SFM-RT'04*, volume 3185 of *LNCS*, pages 1–24, 2004.
3. G. Behrmann, A. David, K. G Larsen, J. Haakansson, P. Pettersson, W. Yi, and M. Hendriks. Uppaal 4.0. In *QEST'06*, pages 125–126, 2006.
4. B. Bérard, B. Bouyer, and A. Petit. Analysing the pgm protocol with UPPAAL. *Int. Journal of Production Research*, 42(14):2773–2791, 2004.
5. B. Berthomieu and M. Menasche. An enumerative approach for analyzing time petri nets. In *IFIP Congress*, pages 41–46, 1983.
6. P. Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.
7. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. KRONOS: a mode-checking tool for real-time systems. In *CAV'98*, volume 1427 of *LNCS*, pages 546–550. Springer, 1998.
8. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS'98*, volume 1384 of *LNCS*, pages 313–329, 1998.
9. D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. Int. Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 197–212. Springer, 1990.
10. A. Gaiser and S. Schwoon. Comparison of algorithms for checking emptiness on büchi automata. In *MEMICS'09*, pages 69–77, 2009.
11. R. Gómez and H. Bowman. Efficient detection of zeno runs in timed automata. In *FORMATS'07*, volume 4763 of *LNCS*, pages 195–210, 2007.
12. K. Havelund, A. Skou, K. Larsen, and K. Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *RTSS'97*, pages 2–13, 1997.

[1] On a 2.4GHz Intel Core 2 Duo MacBook with 2GB of memory.

13. J. J. Jessen, J. I. Rasmussen, K. G. Larsen, and A. David. Guided controller synthesis for climate controller using UPPAAL TiGA. In *FORMATS'07*, volume 4763, pages 227–240. Springer, 2007.
14. S. Schwoon and J. Esparza. A note on on-the-fly verification algorithms. In *TACAS'05*, volume 3440 of *LNCS*, pages 174–190, 2005.
15. S. Tripakis. Verifying progress in timed systems. In *Formal Methods for Real-Time and Probabilistic Systems*, volume 1601 of *LNCS*, pages 299–314, 1999.
16. S. Tripakis. Checking timed büchi emptiness on simulation graphs. *ACM Transactions on Computational Logic*, 10(3), 2009.
17. S. Tripakis, S. Yovine, and A. Bouajjani. Checking timed büchi automata emptiness efficiently. *Formal Methods in System Design*, 26(3):267–292, 2005.