Robust Controller Synthesis in Timed Büchi Automata: A Symbolic Approach

Damien Busatto-Gaston, Benjamin Monmege, Pierre-Alain Reynier, Ocan Sankur

Robust Controller Synthesis in Timed Büchi Automata: A Symbolic Approach*

Damien Busatto-Gaston[1], Benjamin Monmege[1], Pierre-Alain Reynier[1], and Ocan Sankur[2]

[1] Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
[2] Univ Rennes, Inria, CNRS, IRISA, France

Abstract We solve in a purely symbolic way the robust controller synthesis problem in timed automata with Büchi acceptance conditions. The goal of the controller is to play according to an accepting lasso of the automaton, while resisting to timing perturbations chosen by a competing environment. The problem was previously shown to be PSPACE-complete using regions-based techniques, but we provide a first tool solving the problem using zones only, thus more resilient to state-space explosion problems. The key ingredient is the introduction of branching constraint graphs allowing to decide in polynomial time whether a given lasso is robust, and even compute the largest admissible perturbation if it is. We also make an original use of constraint graphs in this context in order to test the inclusion of timed reachability relations, crucial for the termination criterion of our algorithm. Our techniques are illustrated using a case study on the regulation of a train network.

1 Introduction

Timed automata [1] extend finite-state automata with timing constraints, providing an automata-theoretic framework to design, model, verify and synthesise real-time systems. However, the semantics of timed automata is a mathematical idealisation: it assumes that clocks have infinite precision and instantaneous actions. Proving that a timed automaton satisfies a property does not ensure that a real implementation of it also does. This *robustness* issue is a challenging problem for embedded systems [12], and alternative semantics have been proposed, so as to ensure that the verified (or synthesised) behaviour remains correct in presence of small timing perturbations.

We are interested in a fundamental controller synthesis problem in timed automata equipped with a Büchi acceptance condition: it consists in determining whether there exists an accepting infinite execution. Thus, the role of the controller is to choose transitions and delays. This problem has been studied numerously in the exact setting [28,27,19,13,15,17,14]. In the context of robustness, this strategy should be tolerant to small perturbations of the delays. This discards strategies suffering from weaknesses such as Zeno behaviours, or even non-Zeno behaviours requiring infinite precision, as exhibited in [6].

* This work was funded by ANR project Ticktac (ANR-18-CE40-0015).

More formally, the semantics we consider is defined as a game that depends on some parameter δ representing an upper bound on the amplitude of the perturbation [7]. In this game, the controller plays against an antagonistic environment that can perturb each delay using a value chosen in the interval $[-\delta, \delta]$. The case of a fixed value of δ has been shown to be decidable in [7], and also for a related model in [18]. However, these algorithms are based on regions, and as the value of δ may be very different from the constants appearing in the guards of the automaton, do not yield practical algorithms. Moreover, the maximal perturbation is not necessarily known in advance, and could be considered as part of the design process.

The problem we are interested in is *qualitative*: we want to determine whether *there exists* a positive value of δ such that the controller wins the game. It has been proven in [25] that this problem is in PSPACE (and even PSPACE-complete), thus no harder than in the exact setting with no perturbation allowed [1]. However, the algorithm heavily relies on regions, and more precisely on an abstraction that refines the one of regions, namely folded orbit graphs. Hence, it is not at all amenable to implementation.

Our objective is to provide an efficient symbolic algorithm for solving this problem. To this end, we target the use of *zones* instead of regions, as they allow an on-demand partitioning of the state space. Moreover, the algorithm we develop explores the reachable state-space in a *forward* manner. This is known to lead to better performances, as witnessed by the successful tool UPPAAL TIGA based on forward algorithms for solving controller synthesis problems [5].

Our algorithm can be understood as an adaptation to the robustness setting of the standard algorithm for Büchi acceptance in timed automata [17]. This algorithm looks for an accepting lasso using a double depth-first search. A major difficulty consists in checking whether a lasso can be robustly iterated, i.e. whether there exists $\delta > 0$ such that the controller can follow the cycle for an infinite amount of steps while being tolerant to perturbations of amplitude at most δ. The key argument of [25] was the notion of aperiodic folded orbit graph of a path in the region automaton, thus tightly connected to regions. Lifting this notion to zones seems impossible as it makes an important use of the fact that valuations in regions are time-abstract bisimilar, which is not the case for zones.

Our contributions are threefold. First, we provide a polynomial time procedure to decide, given a lasso, whether it can be robustly iterated. This symbolic algorithm relies on a computation of the greatest fixpoint of the operator describing the set of controllable predecessors of a path. In order to provide an argument of termination for this computation, we resort to a new notion of branching constraint graphs, extending the approach used in [16,26] and based on constraint graphs (introduced in [8]) to check iterability of a cycle, without robustness requirements. Second, we show that when considering a lasso, not only can we decide robust iterability, but we can even compute the largest perturbation under which it is controllable. This problem was not known to be decidable before. Finally, we provide a termination criterion for the analysis of lassos. Focusing on zones is not complete: it can be the case that two cycles lead to the same

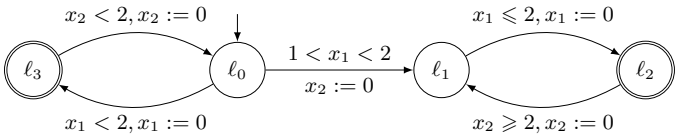

Figure 1. A timed automaton

zones, but one is robustly iterable while the other one is not. Robust iterability crucially depends on the real-time dynamics of the cycle and we prove that it actually only depends on the reachability relation of the path. We provide a polynomial-time algorithm for checking inclusion between reachability relations of paths in timed automata based on constraint graphs. It is worth noticing that all our procedures can be implemented using difference bound matrices, a very efficient data structure used for timed systems. These developments have been integrated in a tool, and we present a case study of a train regulation network illustrating its performances.

Integrating the robustness question in the verification of real-time systems has attracted attention in the community, and the recent works include, for instance, robust model checking for timed automata under clock drifts [23], Lipschitz robustness notions for timed systems [11], quantitative robust synthesis for timed automata [2]. Stability analysis and synthesis of stabilizing controllers in hybrid systems are a closely related topic, see e.g. [21,20].

2 Timed automata: reachability and robustness

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a finite set of clock variables. It is extended with a virtual clock x_0, constantly equal to 0, and we denote by $\mathcal{X}_0$ the set $\mathcal{X} \cup \{x_0\}$. An atomic clock constraint on $\mathcal{X}$ is a formula $x - y \leqslant k$, or $x - y < k$ with $x \neq y \in \mathcal{X}_0$ and $k \in \mathbb{Q}$. A constraint is non-diagonal if one of the two clocks is x_0. We denote by $\mathsf{Guards}(X)$ (respectively, $\mathsf{Guards}_{\mathrm{nd}}(X)$) the set of (clock) constraints (respectively, non-diagonal clock constraints) built as conjunctions of atomic clock constraints (respectively, non-diagonal atomic clock constraints).

A clock valuation ν is an element of $\mathbb{R}_{\geqslant 0}^{\mathcal{X}}$. It is extended to $\mathbb{R}_{\geqslant 0}^{\mathcal{X}_0}$ by letting $\nu(x_0) = 0$. For all $d \in \mathbb{R}_{>0}$, we let $\nu + d$ be the valuation defined by $(\nu + d)(x) = \nu(x) + d$ for all clocks $x \in \mathcal{X}$. If $\mathcal{Y} \subseteq \mathcal{X}$, we also let $\nu[\mathcal{Y} \leftarrow 0]$ be the valuation resetting clocks in $\mathcal{Y}$ to 0, without modifying values of other clocks. A valuation ν satisfies an atomic clock constraint $x - y \bowtie k$ (with $\bowtie \in \{\leqslant, <\}$) if $\nu(x) - \nu(y) \bowtie k$. The satisfaction relation is then extended to clock constraints naturally: the satisfaction of constraint g by a valuation ν is denoted by $\nu \models g$. The set of valuations satisfying a constraint g is denoted by $[\![g]\!]$.

A *timed automaton* is a tuple $\mathcal{A} = (L, \ell_0, E, L_t)$ with L a finite set of locations, $\ell_0 \in L$ an initial location, $E \subseteq L \times \mathsf{Guards}_{\mathrm{nd}}(\mathcal{X}) \times 2^{\mathcal{X}} \times L$ is a finite set of edges, and L_t is a set of accepting locations.

An example of timed automaton is depicted in Figure 1, where the reset of a clock x is denoted by $x := 0$. The semantics of the timed automaton $\mathcal{A}$ is defined as an infinite transition system $[\![\mathcal{A}]\!] = (S, s_0, \to)$. The set S of states of $[\![\mathcal{A}]\!]$ is $L \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$, $s_0 = (\ell_0, \mathbf{0})$. A transition of $[\![\mathcal{A}]\!]$ is of the form $(\ell, \nu) \xrightarrow{e,d} (\ell', \nu')$ with $e = (\ell, g, \mathcal{Y}, \ell') \in E$ and $d \in \mathbb{R}_{>0}$ such that $\nu + d \models g$ and $\nu' = (\nu + d)[\mathcal{Y} \leftarrow 0]$. We call *path* a possible finite sequence of edges in the timed automaton. The *reachability relation* of a path ρ, denoted by $\mathsf{Reach}(\rho)$ is the set of pairs (ν, ν') such that there is a sequence of transitions of $[\![\mathcal{A}]\!]$ starting from (ℓ, ν), ending in (ℓ', ν') and that follows ρ in order as the edges of the timed automaton. A *run* of $\mathcal{A}$ is an infinite sequence of transitions of $[\![\mathcal{A}]\!]$ starting from s_0. We are interested in Büchi objectives. Therefore, a run is accepting if there exists a final location $\ell_t \in L_t$ that the run visits infinitely often.

As done classically, we assume that every clock is bounded in $\mathcal{A}$ by a constant M, that is we only consider the previous infinite transition system over the subset $L \times [0, M]^{\mathcal{X}}$ of states.

We study the robustness problem introduced in [25], that is stated in terms of games where a controller fights against an environment. After a prefix of a run, the controller will have the capability to choose delays and transitions to fire, whereas the environment perturbs the delays chosen by the controller with a small parameter $\delta > 0$. The aim of the controller will be to find a strategy so that, no matter how the environment plays, he is ensured to generate an infinite run satisfying the Büchi condition. Formally, given a timed automaton $\mathcal{A} = (L, \ell_0, E, L_t)$ and $\delta > 0$, the perturbation game is a two-player turn-based game $\mathcal{G}_\delta(\mathcal{A})$ between a controller and an environment. Its state space is partitioned into $S_C \uplus S_E$ where $S_C = L \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ belongs to the controller, and $S_E = L \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0} \times \mathbb{R}_{>0} \times E$ to the environment. The initial state is $(\ell_0, \mathbf{0}) \in S_C$. From each state $(\ell, \nu) \in S_C$, there is a transition to $(\ell, \nu, d, e) \in S_E$ with $e = (\ell, g, \mathcal{Y}, \ell') \in E$ whenever $d > \delta$, and $\nu + d + \varepsilon \models g$ for all $\varepsilon \in [-\delta, \delta]$. Then, from each state $(\ell, \nu, d, (\ell, g, \mathcal{Y}, \ell')) \in S_E$, there is a transition to $(\ell', (\nu + d + \varepsilon)[r \leftarrow 0]) \in S_C$ for all $\varepsilon \in [-\delta, \delta]$. A play of $\mathcal{G}_\delta(\mathcal{A})$ is a finite or infinite path $q_0 \xrightarrow{t_1} q_1 \xrightarrow{t_2} q_2 \cdots$ where $q_0 = (\ell_0, 0)$ and t_i is a transition from state q_{i-1} to q_i, for all $i > 0$. It is said to be maximal if it is infinite or can not be extended with any transition.

A strategy for the controller is a function σ_{Con} mapping each non-maximal play ending in some $(\ell, \nu) \in S_C$ to a pair (d, e) where $d > 0$ and $e \in E$ such that there is a transition from (ℓ, ν) to (ℓ, ν, d, e). A strategy for the environment is a function σ_{Env} mapping each finite play ending in (ℓ, ν, d, e) to a state (ℓ', ν') related by a transition. A play gives rise to a unique run of $[\![\mathcal{A}]\!]$ by only keeping states in V_C. For a pair of strategies $(\sigma_{\mathsf{Con}}, \sigma_{\mathsf{Env}})$, we let $\mathsf{play}^\delta_{\mathcal{A}}(\sigma_{\mathsf{Con}}, \sigma_{\mathsf{Env}})$ denote the run associated with the unique maximal play of $\mathcal{G}_\delta(\mathcal{A})$ that follows the strategies. Controller's strategy σ_{Con} is winning (with respect to the Büchi objective L_t) if for all strategies σ_{Env} of the environment, $\mathsf{play}^\delta_{\mathcal{A}}(\sigma_{\mathsf{Con}}, \sigma_{\mathsf{Env}})$ is infinite and visits infinitely often some location of L_t. The *parametrised robust controller synthesis problem* asks, given a timed automaton $\mathcal{A}$, whether there exists $\delta > 0$ such that the controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$.

Example 1. The controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$, with $\mathcal{A}$ the automaton of Figure 1, for all possible values of $\delta < 1/2$. Indeed, he can follow the cycle $\ell_0 \to \ell_3 \to \ell_0$ by always picking time delay $1/2$ so that, when arriving in ℓ_3 (resp. ℓ_0) after the perturbation of the environment, clock x_2 (resp. x_1) has a valuation in $[1/2 - \delta, 1/2 + \delta]$. Therefore, he can play forever following this memoryless strategy. For $\delta \geq 1/2$, the environment can enforce reaching ℓ_3 with a value for x_2 at least equal to 1. The guard $x_2 < 2$ of the next transition to ℓ_0 cannot be guaranteed, and therefore the controller cannot win $\mathcal{G}_\delta(\mathcal{A})$. In [25], it is shown that the cycle around ℓ_2 does not provide a winning strategy for the controller for any value of $\delta > 0$ since perturbations accumulate so that the controller can only play it a finite number of times in the worst case.

By [25], the parametrised robust controller synthesis problem is known to be PSPACE-complete. Their solution is based on the region automaton of $\mathcal{A}$. We are seeking for a more practical solution using zones. A zone Z over $\mathcal{X}$ is a convex subset of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ defined as the set of valuations satisfying a clock constraint g, i.e. $Z = [\![g]\!]$. Zones can be encoded into *difference-bound matrices (DBM)*, that are $|\mathcal{X}_0| \times |\mathcal{X}_0|$-matrices over $(\mathbb{R} \times \{<, \leqslant\}) \cup \{(\infty, <)\}$. We adopt the following notation: for a DBM M, we write $M = (\mathsf{M}, \prec^M)$, where M is the matrix made of the first components, with elements in $\mathbb{R} \cup \{\infty\}$, while $\prec^M$ is the matrix of the second components, with elements in $\{<, \leqslant\}$. A DBM M naturally represents a zone (which we abusively write M as well), defined as the set of valuations ν such that, for all $x, y \in \mathcal{X}_0$, $\nu(x) - \nu(y) \prec^M_{x,y} \mathsf{M}_{x,y}$ (where $\nu(x_0) = 0$). Coefficients of a DBM are thus pairs $(\prec, c)$. As usual, these can be compared: $(\prec, c)$ is less than $(\prec', c')$ (denoted by $(\prec, c) < (\prec', c')$) whenever $c < c'$ or ($c = c'$, $\prec = <$ and $\prec' = \leqslant$). Moreover, these coefficients can be added: $(\prec, c) + (\prec', c')$ is the pair $(\prec'', c + c')$ with $\prec'' = \leqslant$ if $\prec = \prec' = \leqslant$ and $\prec'' = <$ otherwise.

DBMs were introduced in [4,10] for analyzing timed automata; we refer to [3] for details. Standard operations used to explore the state space of timed automata have been defined on DBMs: intersection is written $M \cap N$, $\mathsf{Pretime}_{>t}(M)$ is the set of valuations such that a time delay of more than t time units leads to the zone M, $\mathsf{Unreset}_R(M)$ is the set of valuations that end in M when the clocks in R are reset. From a robustness perspective, we also consider the operator $\mathsf{shrink}_{[-\delta,\delta]}(M)$ defined as the set of valuations ν such that $\nu + [-\delta, \delta] \subseteq M$ introduced in [24]. Given a DBM M and a rational number δ, all these operations can be effectively computed in time cubic in $|\mathcal{X}|$.

3 Reachability relation of a path

Before treating the robustness issues, we start by designing a symbolic (i.e. zone-based) approach to describe and compare the reachability relations of paths in timed automata. This will be crucial subsequently to design a termination criterion in the state space exploration of our robustness-checking algorithm. Solving the inclusion of reachability relations in a symbolic manner has independent interest and can have other applications.

The reachability relation $\mathsf{Reach}(\rho)$ of a path ρ, is a subset of $\mathbb{R}_{\geqslant 0}^{\mathcal{X}\cup\mathcal{X}'}$ where $\mathcal{X}'$ are primed versions of the clocks, such that each $(\nu,\nu') \in \mathsf{Reach}(\rho)$ iff there is a run from valuation ν to valuation ν' following ρ. Unfortunately, reachability relations $\mathsf{Reach}(\rho)$ are not zones in general, that is, they cannot be represented using only difference constraints. In fact, we shall see shortly that constraints of the form $x - y + z - u \leqslant c$ also appear, as already observed in [22]. We thus cannot rely directly on the traditional difference bound matrices (DBMs) used to represent zones. We instead rely on the constraint graphs that were introduced in [8], and explored in [16] for the parametric case (the latter work considers enlarged constraints, and not shrunk ones as we study here). Our contribution is to use these graphs to obtain a syntactic check of inclusion of the according reachability relations.

Constraint graphs. Rather than considering the values of the clocks in $\mathcal{X}$, this data structure considers the date X_i of the latest reset of the clock x_i, and uses a new variable τ denoting the global timestamp. Note that the clock values can be recovered easily since $X_i = \tau - x_i$. For the extra clock x_0, we introduce variable X_0 equal to the global timestamp τ (since x_0 must remain equal to 0). A constraint graph defining a zone is a weighted graph whose nodes are $X = \{X_0, X_1, \ldots, X_n\}$. Constraints on clocks are represented by weights on edges in the graph: a constraint $X - Y \prec c$ is represented by an edge from X to Y weighted by $(\prec, c)$, with $\prec \in \{\leqslant, <\}$ and $c \in \mathbf{Q}$. Weights in the graph are thus pairs of the form $(\prec, c)$. Therefore, we can compute shortest weights between two vertices of a weighted graph. A cycle is said to be negative if it has weight at most $(<, 0)$, i.e. $(<, 0)$ or $(\prec, c)$ with $c < 0$.

Encoding paths. Constraint graphs can also encode tuples of valuations seen along a path. To encode a k-step computation, we make $k+1$ copies of the nodes, that is, $X^i = \{X_0^i, X_1^i, \ldots, X_n^i\}$ for $i \in \{1, \ldots, k+1\}$. These copies are also called *layers*. Let us first consider an example on the path ρ consisting of the edge from ℓ_1 to ℓ_2, and the edge from ℓ_2 to ℓ_1, in the timed automaton of Figure 1. The constraint graph G_ρ is depicted in Figure 2: in our diagrams of constraint graphs, the absence of labels on an edge means $(\leqslant, 0)$, and we depict with an edge with arrows on both ends the presence of an edge in both directions. The graph has five columns, each containing copies of the variables for that step: they represent the valuations before the first edge, after the first time elapse, after the first reset, after the second time elapse and after the second reset. In general now, each elementary operation can be described by a constraint graph with two layers (X_i) (before) and (X_i') (after).

- The operation $\mathsf{Pretime}_{>t}$ is described by the constraint graph $G_{\mathsf{time}}^{>t}$ with edges $X_i \to X_0$, $X_i \leftrightarrow X_i'$ for $i > 0$, and $X_0 \xrightarrow{(<,-t)} X_0'$. Figure 2 contains two occurrences of $G_{\mathsf{time}}^{>0}$: we always represent with dashed arrows edges that are labelled by $(<, c)$, and plain arrows edges that are labelled with $(\leqslant, c)$; the absence of an edge means that it is labelled with $(<, \infty)$.
- The operation $g \cap \mathsf{Unreset}_{\mathcal{Y}}(\cdot)$, to test a guard g and reset the clocks in $\mathcal{Y}$, is described by the constraint graph $G_{\mathsf{edge}}^{g,\mathcal{Y}}$ with edges $X_0 \leftrightarrow X_0'$ (meaning

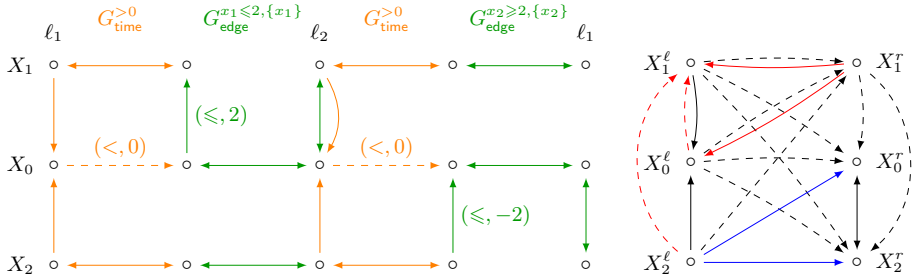

Figure 2. On the left, the constraint graph of the path $\ell_1 \xrightarrow{x_1\leqslant 2, x_1:=0} \ell_2 \xrightarrow{x_2\geqslant 2, x_2:=0} \ell_1$. On the right, its normalised version: dashed edges have weight $(<, .)$, plain edges have weight $(\leqslant, .)$, black edges have weight $(., 0)$, red edges have weight $(., 2)$ and blue edges have weight $(., -2)$.

that the time does not elapse), $X_i \leftrightarrow X_i'$ for i such that clock $x_i \notin \mathcal{Y}$, and $X_i' \leftrightarrow X_0'$ for i such that clock $x_i \in \mathcal{Y}$, and for all clock constraint $x_i - x_j \prec c$ appearing in g, an edge from X_j to X_i labelled by $(\prec, c)$ (since it encodes the fact that $(\tau - X_i) - (\tau - X_j) = X_j - X_i \prec c$). In Figure 2, we have first $G_{\text{edge}}^{x_1\leqslant 2,\{x_1\}}$, and then $G_{\text{edge}}^{x_2\geqslant 2,\{x_2\}}$.

Constraint graphs can be stacked one after the other to obtain the constraint graph of an edge e, and then of a path ρ, that we denote by G_ρ. In the resulting graph, there is one leftmost layer of vertices $(X_i^\ell)_i$ and one rightmost one $(X_i^r)_i$ representing the situation before and after the firing of the path ρ. Once this graph is constructed, the intermediary levels can be eliminated after replacing each edge between the nodes of $X^\ell \cup X^r$ by the shortest path in the graph. This phase is hereafter called *normalisation* of the constraint graph. The normalised version of the constraint graph of Figure 2 is depicted on its right.

From constraint graphs to reachability relations. From a logical point of view, the elimination of intermediary layers reflects an elimination of quantifiers in a formula of the first-order theory of real numbers. At the end, we obtain a set of constraints of the form $X_i^k - X_j^{k'} \prec c$ with $k, k' \in \{\ell, r\}$. These constraints do not reflect uniquely the reachability relation $\mathsf{Reach}(\rho)$, in the sense that it is possible that $\mathsf{Reach}(\rho_1) = \mathsf{Reach}(\rho_2)$ but the normalised versions of G_{ρ_1} and G_{ρ_2} are different. For example, if we consider the path ρ^2 obtained by repeating the cycle ρ between ℓ_1 and ℓ_2, the reachability relation does not change ($\mathsf{Reach}(\rho^2) = \mathsf{Reach}(\rho)$), but the normalised constraint graph does ($G_{\rho^2} \neq G_{\rho^1}$): all labels $(\leqslant, 2)$ of the red dotted edges from the rightmost layer to the leftmost layer become $(\leqslant, 4)$, and the labels $(\leqslant, -2)$ of the dashed blue edges become $(\leqslant, -4)$.

We solve this issue by jumping back from variables X_i^k to the clock valuations. Indeed, in terms of clock valuations ν^ℓ and ν^r before and after the path, the constraint $X_i^k - X_j^{k'} \prec c$ (for $k, k' \in \{l, r\}$) rewrites as $(\tau^k - \nu^k(x_i)) - (\tau^{k'} - \nu^{k'}(x_j)) \prec c$, where τ^ℓ is the global timestamp before firing ρ and τ^r the one after. When $k = k'$, variables τ^ℓ and τ^r disappear, leaving a constraint of the form

$\nu^k(x_j) - \nu^k(x_i) \prec c$. When $k \neq k'$, we can rewrite the constraint as $\tau^k - \tau^{k'} \prec \nu^k(x_i) - \nu^{k'}(x_j) + c$. We therefore obtain upper and lower bounds on the value of $\tau^r - \tau^\ell$, allowing us to eliminate $\tau^r - \tau^\ell$ considered as a single variable. We therefore obtain in fine a formula mixing constraints of the form

- $\nu^k(x_a) - \nu^k(x_b) \prec p$, with $k \in \{\ell, r\}$, $a \neq b$, and we define $\gamma^k_{a,b} = (\prec, p)$;
- $\nu^\ell(x_a) - \nu^\ell(x_b) + \nu^r(x_c) - \nu^r(x_d) \prec p$, with $a \neq b$ and $c \neq d$, and we define $\gamma_{a,b,c,d} = (\prec, p)$. This constraint can appear in two ways: either from $\nu^r(x_c) - \nu^\ell(x_b) + p_1 \prec_1 \tau^r - \tau^l \prec_2 \nu^l(x_a) - \nu^r(x_d) + p_2$ by eliminating $\tau^r - \tau^l$, or by adding the two constraints of the form $\nu^l(x_a) - \nu^l(x_b) \prec_1 p_1$ and $\nu^r(x_c) - \nu^r(x_d) \prec_2 p_2$. Thus, $\gamma_{a,b,c,d}$ is obtained as the minimum of the two constraints obtained in this manner. In other terms, in the constraint graph, this constraint is the minimal weight between the sum of the weights of the edges (X^r_d, X^l_a) and (X^l_b, X^r_c), and the sum of the weights of the edges (X^l_b, X^l_a) and (X^r_d, X^r_c). For example, in the path in Figure 2, we have $\gamma_{0,1,0,2} = (\leqslant, 0)$ since the two constraints are $(\leqslant, 0)$ and $(<, \infty)$, whereas $\gamma_{1,2,2,1} = (\leqslant, 0)$ because the two constraints are $(<, 2)$ and $(\leqslant, 0)$.

Let $\varphi(G)$ be the conjunction of such constraints obtained from a constraint graph G once normalised: this is a quantifier-free formula of the additive theory of reals. We obtain the following property whose proof mimics the one for proving the normalisation of DBMs (and can be derived from the developments of [8]).

Lemma 1. *Let ρ be a path in a timed automaton. If G_ρ contains a negative cycle, then $\mathsf{Reach}(\rho) = \emptyset$. Otherwise, $\mathsf{Reach}(\rho)$ is the set of pairs of valuations (ν^ℓ, ν^r) that satisfy the formula $\varphi(G_\rho)$.*

Checking inclusion. For a path ρ, we regroup the pairs $(\gamma^l_{a,b})$, $(\gamma^r_{a,b})$ and $(\gamma_{a,b,c,d})$ above in a single vector Γ^ρ. We extend the comparison relation $<$ to these vectors by applying it componentwise. These vectors can be used to check equality or inclusion of reachability relations in time $O(|X|^4)$:

Theorem 1. *Let ρ and ρ' be paths in a timed automaton such that $\mathsf{Reach}(\rho)$ and $\mathsf{Reach}(\rho')$ are non empty. Then $\mathsf{Reach}(\rho) \subseteq \mathsf{Reach}(\rho')$ if and only if $\Gamma^\rho \leqslant \Gamma^{\rho'}$.*

Notice that we do not need to check equivalence or implication of formulas $\varphi(G_\rho)$ and $\varphi(G_{\rho'})$, but simply check syntactically constants appearing in these formulas. Moreover, these constants can be stored in usual DBMs on $2 \times |\mathcal{X}_0|$ clocks, allowing for reusability of classical DBM libraries. For the constraint graph in Figure 2, we have seen that $G_{\rho^2} \neq G_{\rho^1}$, even if $\mathsf{Reach}(\rho^2) = \mathsf{Reach}(\rho)$. However, we can check that $\varphi(G_{\rho^2}) = \varphi(G_\rho)$ as expected.

Computation of Pre and Post. By Lemma 1 and the construction of constraint graphs, one can easily compute $\mathsf{Pre}_\rho(Z) = \{\nu \mid \exists \nu' \in Z \ ((\ell, \nu), (\ell', \nu')) \in \mathsf{Reach}(\rho)\}$ for a given path ρ and zone Z (see [8,16]). In fact, consider the normalised constraint graph G_ρ on nodes $X^\ell \cup X^r$. To compute $\mathsf{Pre}_\rho(Z)$, one just needs to add the constraints of Z on X^r. This is done by replacing each edge $X^r_i \xrightarrow{w} X^r_j$ by $X^r_i \xrightarrow{\min(Z_{j,i}, w)} X^r_j$ where $Z_{j,i} = (\prec, p)$ defines the constraint

of Z on $x_j - x_i$. Then, the normalisation of the graph describes the reachability relation along path ρ ending in zone Z. Furthermore, projecting the constraints to X^ℓ yields $\mathsf{Pre}_\rho(Z)$: this can be obtained by gathering all constraints on pairs of nodes of X^ℓ. A reachability relation can thus be seen as a function assigning to each zone Z its image by ρ. One can symmetrically compute the successor $\mathsf{Post}_\rho(Z) = \{\nu' \mid \exists \nu \in Z\ ((\ell,\nu),(\ell',\nu')) \in \mathsf{Reach}(\rho)\}$ by constraining the nodes X^ℓ and projecting to X^r.

4 Robust iterability of a lasso

In this section, we study the perturbation game $\mathcal{G}_\delta(\mathcal{A})$ between the two players (controller and environment), as defined in Section 2, when the timed automaton $\mathcal{A}$ is restricted to a fixed *lasso* $\rho_1\rho_2$, i.e. ρ_1 is a path from ℓ_0 to some accepting location ℓ_t, and ρ_2 a cyclic path around ℓ_t. This implies that the controller does not have the choice of the transitions, but only of the delays. We will consider different settings, in which δ is fixed or not.

Controllable predecessors and their greatest fixpoints. Consider an edge $e = (\ell, g, R, \ell')$. For any set $Z \subseteq \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$, we define the *controllable predecessors of* Z as follows: $\mathsf{CPre}^\delta_e(Z) = \mathsf{Pretime}_{>\delta}(\mathsf{shrink}_{[-\delta,\delta]}(g \cap \mathsf{Unreset}_R(Z)))$. Intuitively, $\mathsf{CPre}^\delta_e(Z)$ is the set of valuations from which the controller can ensure reaching Z in one step, following the edge e, no matter of the perturbations of amplitude at most δ of the environment. In fact, it can delay in $\mathsf{shrink}_{[-\delta,\delta]}(g \cap \mathsf{Unreset}_R(Z))$ with a delay of at least δ, where under any perturbation in $[-\delta,\delta]$, the valuation satisfies the guard, and it ends, after reset, in Z. Results of [24] show that this operator can be computed in cubic time with respect to the number of clocks. We extend this operator to a path ρ by composition, denoted it by $\mathsf{CPre}^\delta_\rho$. Note that $\mathsf{CPre}^0_\rho = \mathsf{Pre}_\rho$ is the usual predecessor operator without perturbation.

This operator is monotone, hence its greatest fixpoint $\nu X\ \mathsf{CPre}^\delta_\rho(X)$ is well-defined, equal to $\bigcap_{i\geqslant 0} \mathsf{CPre}^\delta_{\rho^i}(\top)$: it corresponds to the valuations from which the controller can guarantee to loop forever along the path ρ. By definition of the game $\mathcal{G}_\delta(\mathcal{A})$ where $\mathcal{A}$ is restricted to the lasso $\rho_1\rho_2$, the controller wins the game if and only if $\mathbf{0} \in \mathsf{CPre}^\delta_{\rho_1}(\nu X\ \mathsf{CPre}^\delta_{\rho_2}(X))$. As a consequence, our problem reduces to the computation of this greatest fixpoint.

Branching constraint graphs. We consider first a fixed (rational) value of the parameter δ, and are interested in the computation of the greatest fixpoint $\nu X\ \mathsf{CPre}^\delta_{\rho_2}(X)$. In [16], constraints graphs were used to provide a termination criterion allowing to compute the greatest fixpoint of the classical predecessor operator CPre^0_ρ. We generalize this approach to deal with the operator $\mathsf{CPre}^\delta_\rho$ and to this end, we need to generalize constraint graphs so as to encode it. Unfortunately, the operator $\mathsf{shrink}_{[-\delta,\delta]}$ cannot be encoded in a constraint graph. Intuitively, this comes from the fact that a constraint graph represents a relation between valuations, while there is no such relation associated with the $\mathsf{CPre}^\delta_\rho$ operator. Instead, we introduce *branching constraint graphs*, that will faithfully represent the $\mathsf{CPre}^\delta_\rho$ operator: unlike constraint graphs introduced so far that

have a left layer and a right layer of variables, a branching constraint graph has still a single left layer but several right layers.

We first define a branching constraint graph $G^{\delta}_{\mathsf{shrink}}$ associated with the operator $\mathsf{shrink}_{[-\delta,\delta]}$ as follows. Its set of vertices is composed of three copies of the $\{X_0, X_1, \ldots, X_n\}$, denoted by primed, unprimed and doubly primed versions. Edges are defined so as to encode the following constraints : $X'_i = X_i$ and $X''_i = X_i$ for every $i \neq 0$, and $X'_0 = X_0 + \delta$ and $X''_0 = X_0 - \delta$. An instance of this graph can be found in several occurrences in Figure 3.

Proposition 1. *Let Z be a zone and $G^{\delta}_{\mathsf{shrink}}(Z)$ be the graph obtained from $G^{\delta}_{\mathsf{shrink}}$ by adding on primed and doubly primed vertices the constraints defining Z (as for $\mathsf{Pre}_\rho(Z)$ in the end of Section 3). Then the constraint on unprimed vertices obtained from the shortest paths in $G^{\delta}_{\mathsf{shrink}}(Z)$ is equivalent to $\mathsf{shrink}_{[-\delta,\delta]}(Z)$.*

Proof. Given a zone Z and a real number d, we define $Z + d = \{\nu + d \mid \nu \in Z\}$. One easily observes that $\mathsf{shrink}_{[-\delta,\delta]}(Z) = (Z + \delta) \cap (Z - \delta)$. The result follows from the observation that taking two distinct copies of vertices, and considering shortest paths allows one to encode the intersection. □

Then, for all edges $e = (\ell, g, R, \ell')$, we define the branching constraint graph G^{δ}_{e} as the graph obtained by stacking (in this order) the branching constraint graph $G^{>\delta}_{\mathsf{time}}$, $G^{\delta}_{\mathsf{shrink}}$ and $G^{g,\mathcal{Y}}_{\mathsf{edge}}$. Note that two copies of the graph $G^{g,\mathcal{Y}}_{\mathsf{edge}}$ are needed, to be connected to the two sets of vertices that are on the right of the graph $G^{\delta}_{\mathsf{shrink}}$. This definition is extended in the expected way to a finite path ρ, yielding the graph G^{δ}_{ρ}. In this graph, there is a single set of vertices on the left, and $2^{|\rho|}$ sets of vertices on the right. As a direct consequence of the previous results on the constraint graphs for time elapse, shrinking and guard/reset, one obtains:

Proposition 2. *Let Z be a zone and ρ be a path. We let $G^{\delta}_{\rho}(Z)$ be the graph obtained from G^{δ}_{ρ} by adding on every set of right vertices the constraints defining Z. Then the constraint on the left layer of vertices obtained from the shortest paths in $G^{\delta}_{\rho}(Z)$ is equivalent to $\mathsf{CPre}^{\delta}_{\rho}(Z)$.*

An example of the graph $G^{\delta}_{\rho}(Z)$ for $\rho = e_1 e_2$, edges considered in Figure 2, is depicted in Figure 3 (on the left).

We are now ready to prove the following result, generalisation of [16, Lemma 2], that will allow us to compute the greatest fixpoint of the operator $\mathsf{CPre}^{\delta}_{\rho}$:

Proposition 3. *Let ρ be a path and δ be a non-negative rational number. We let $N = |\mathcal{X}_0|^2$. If $\mathsf{CPre}^{\delta}_{\rho^{2N+1}}(\top) \subsetneq \mathsf{CPre}^{\delta}_{\rho^{2N}}(\top)$, then $\nu X\ \mathsf{CPre}^{\delta}_{\rho}(X) = \emptyset$.*

Proof. Assume $\mathsf{CPre}^{\delta}_{\rho^{2N+1}}(\top) \subsetneq \mathsf{CPre}^{\delta}_{\rho^{2N}}(\top)$ and consider the zones $\mathsf{CPre}^{\delta}_{\rho^{N+1}}(\top)$ (represented by the DBM M_1) and $\mathsf{CPre}^{\delta}_{\rho^{N}}(\top)$ (represented by the DBM M_2). We have $M_1 \subsetneq M_2$, as otherwise the fixpoint would have already been reached after N steps. By Proposition 2, the zone corresponding to M_1 is associated with shortest paths between vertices on the left in the graph $G^{\delta}_{\rho^{N+1}}$. In the sequel, given a path r in this graph, $w(r)$ denotes its weight. We distinguish two cases:

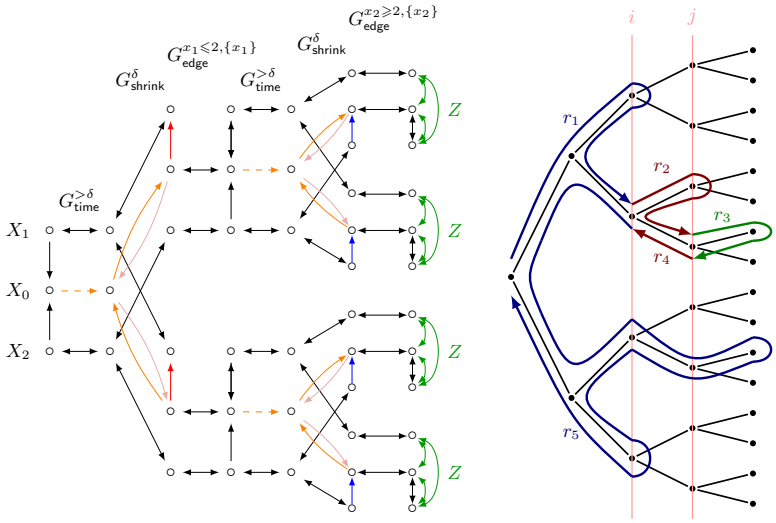

Figure 3. On the left, the branching constraint graph $G^{\delta}_{e_1 e_2}$ encoding the operator $\mathsf{CPre}^{\delta}_{e_1 e_2}$, where e_1 and e_2 refer to edges considered in Figure 2. Dashed edges have weight $(<, .)$, plain edges have weight $(\leqslant, .)$. Black edges (resp. orange edges, pink edges, red edges, blue edges) are labelled by $(., 0)$ (resp. $(., -\delta)$, $(., \delta)$, $(., 2)$,$(., -2)$). On the right, a decomposition of a path in a branching constraint graph G^{δ}_{ρ}.

Case 1: $M_1 \subsetneq M_2$ because of the rational coefficients. Then, there exists an entry $(x, y) \in \mathcal{X}_0^2$ such that $\mathsf{M}_1[x, y] < \mathsf{M}_2[x, y]$. The value $M_1[x, y]$ is thus associated with a shortest path between vertices X and Y in $G^{\delta}_{\rho^{N+1}}$. We fix a shortest path of minimal length, and denote it by r. As the entry is strictly smaller than in M_2, this shortest path should reach the last copy of the graph G^{δ}_{ρ}. This path can be interpreted as a traversal of the binary tree of depth $|\mathcal{X}_0|^2 + 1$, reaching at least one leaf. We can prove that this entails that there exists a pair of clocks $(u, v) \in \mathcal{X}_0^2$ appearing at two levels $i < j$ of this tree, and a decomposition $r = r_1 r_2 r_3 r_4 r_5$ of the path, such that $w(r_2) + w(r_4) = (\prec, d)$ with $d < 0$ (Property (†)). In addition, in this decomposition, r_3 is included in subgraphs of levels $k \geq j$, and the pair of paths (r_2, r_4) is called a *return path*, following the terminology of [16]. This decomposition is depicted in Figure 3 (on the right). Intuitively, the property (†) follows from the fact that as r_3 is included in subgraphs of levels $k \geq j$, and because the final zone (on the right) is the zone $\top$ which adds no edges, the concatenation $r' = r_1 r_3 r_5$ is also a valid path from X to Y in $G^{\delta}_{\rho^{N+1}}$, and is shorter than r. We conclude using the fact that r has been chosen as a shortest path of minimal weight.

Property (†) allows us to prove that the greatest fixpoint is empty. Indeed, by considering iterations of ρ, one can repeat the return path associated with (r_2, r_4) and obtain paths from X to Y whose weights diverge towards $-\infty$.

Case 2: $M_1 \subsetneq M_2$ because of the ordering coefficients. We claim that this case cannot occur. Indeed, one can show that the constants will not evolve anymore

after the Nth iteration of the fixpoint: the coefficients can only decrease by changing from a non-strict inequality $(\leq, c)$ to a strict one $(<, c)$. This propagation of strict inequalities is performed in at most $|\mathcal{X}_0|^2$ additional steps, thus we have $\mathsf{CPre}^{\delta}_{\rho^{2N+1}}(\top) = \mathsf{CPre}^{\delta}_{\rho^{2N}}(\top)$, yielding a contradiction. □

Compared to the result of [16], the number of iterations needed before convergence grows from $|\mathcal{X}_0|^2$ to $2|\mathcal{X}_0|^2$: this is due to the presence of strict and non-strict inequalities, not considered in [16]. With the help of branching constraint graphs, we have thus shown that the greatest fixpoint can be computed in finite time: this can then be done directly with computations on zones (and not on branching constraint graphs).

Proposition 4. *Given a path ρ and a rational number δ, the greatest fixpoint $\nu X\ \mathsf{CPre}^{\delta}_{\rho}(X)$ can be computed in time polynomial in $|\mathcal{X}|$ and $|\rho|$. As a consequence, one can decide whether the controller has a strategy along a lasso $\rho_1\rho_2$ in $\mathcal{G}_\delta(\mathcal{A})$ in time polynomial in $|\mathcal{X}|$ and $|\rho_1\rho_2|$.*

Solving the robust controller synthesis problem for a lasso. We have shown how to decide whether the controller has a winning strategy for a fixed rational value of δ. We now aim at deciding whether there exists a positive value of δ for which the controller wins the game $\mathcal{G}_\delta(\mathcal{A})$ (where $\mathcal{A}$ is restricted to a lasso $\rho_1\rho_2$). To this end, we will use a parametrised extension of DBMs, namely *shrunk DBMs*, that were introduced in [24] in order to study the parametrised state space of timed automata. Intuitively, our goal is to express *shrinkings* of guards, e.g. sets of states satisfying constraints of the form $g = 1 + \delta < x < 2 - \delta \wedge 2\delta < y$, where δ is a parameter to be chosen. Formally, a shrunk DBM is a pair (M, P), where M is a DBM, and P is a nonnegative integer matrix called a *shrinking matrix*. This pair represents the set of valuations defined by the DBM $M - \delta P$, for any given $\delta > 0$. Considering the example g, M is the guard g obtained by setting $\delta = 0$, and P is made of the integer multipliers of δ. We adopt the following notation: when we write a statement involving a shrunk DBM (M, P), we mean that for some $\delta_0 > 0$, the statement holds for $M - \delta P$ for all $\delta \in (0, \delta_0]$. For instance, $(M, P) = \mathsf{Pretime}_{>\delta}((N, Q))$ means that $M - \delta P = \mathsf{Pretime}_{>\delta}(N - \delta Q)$ for all small enough $\delta > 0$. Shrunk DBMs are closed under standard operations on zones, and as a consequence, the CPre operator can be computed on shrunk DBMs:

Lemma 2 ([25]). *Let $e = (\ell, g, R, \ell')$ be an edge and (M, P) be a shrunk DBM. Then, there exists a shrunk DBM (N, Q), that we can compute in polynomial time, such that $(N, Q) = \mathsf{CPre}^{\delta}_{e}((M, P))$.*

Proposition 5. *Given a path ρ, one can compute a shrunk DBM (M, P) equal to the greatest fixpoint of the operator $\mathsf{CPre}^{\delta}_{\rho}$. As a consequence, one can solve the parametrised robust controller synthesis problem for a given lasso in time complexity polynomial in the number of clocks and in the length of the lasso.*

Proof. The bound $2|\mathcal{X}_0|^2$ identified previously does not depend on the value of δ. Hence the algorithm for computing a shrunk DBM representing the greatest fixpoint proceeds as follows. It computes symbolically, using shrunk DBMs, the

$2|\mathcal{X}_0|^2$-th and $2|\mathcal{X}_0|^2+1$-th iterations of the operator $\mathsf{CPre}^\delta_\rho$, from the zone $\top$. By monotonicity, the $2|\mathcal{X}_0|^2+1$-th iteration is included in the $2|\mathcal{X}_0|^2$-th. If the two shrunk DBMs are equal, then they are also equal to the greatest fixpoint. Otherwise, the greatest fixpoint is empty. To decide the robust controller synthesis problem for a given lasso, one first computes a shrunk DBM representing the greatest fixpoint associated with ρ_2 and, if not empty, one computes a new shrunk DBM by applying to it the operator $\mathsf{CPre}^\delta_{\rho_1}$. Then, one checks whether the valuation $\mathbf{0}$ belongs to the resulting shrunk DBM. □

Computing the largest admissible perturbation. We say that a perturbation δ is *admissible* if the controller wins the game $\mathcal{G}_\delta(\mathcal{A})$. The parametrised robust controller synthesis problem, solved before just for a lasso, aims at deciding whether *there exists* a positive admissible perturbation. A more ambitious problem consists in determining the *largest admissible* perturbation.

The previous algorithm performs a bounded $(2|\mathcal{X}_0|^2)$ number of computations of the $\mathsf{CPre}^\delta_\rho$ operator. Instead of focusing on arbitrarily small values using shrunk DBMs as we did previously, we must perform a computation for all values of δ. To do so, we consider an extension of the (shrunk) DBMs in which each entry of the matrix (which thus represents a clock constraint) is a piecewise affine function of δ. One can observe that all the operations involved in the computation of the $\mathsf{CPre}^\delta_\rho$ operator can be performed symbolically w.r.t. δ using piecewise affine functions. As a consequence, we obtain the following new result:

Proposition 6. *We can compute the largest admissible perturbation of a lasso.*

Proof. Let $\rho_1\rho_2$ be a lasso. One first computes a symbolic representation, valid for all values of δ, of the greatest fixpoint of $\mathsf{CPre}^\delta_{\rho_2}$. To do so, one computes the $2|\mathcal{X}_0|^2$-th and $2|\mathcal{X}_0|^2+1$-th iterations of this operator, from the zone $\top$. We denote them by M_1 and M_2 respectively. By monotonicity, the inclusion $M_1(\delta) \subseteq M_2(\delta)$ holds for every $\delta \geq 0$. In addition, both M_1 and M_2 are decreasing w.r.t. δ, thus one can identify the value $\delta_0 = \inf\{\delta \geq 0 \mid M_1(\delta) \subsetneq M_2(\delta)\}$. Then, the greatest fixpoint is equal to M_1 for $\delta < \delta_0$, and to the emptyset for δ at least δ_0. As a second step, one applies the operator CPre_{ρ_1} to the greatest fixpoint. We denote the result by M. To conclude, one can then compute and return the value $\sup\{\delta \in [0, \delta_0[\mid \mathbf{0} \in M(\delta)\}$ of maximal perturbation. □

5 Synthesis of robust controllers

We are now ready to solve the parametrised robust controller synthesis problem, that is to find, if it exists, a lasso $\rho_1\rho_2$ and a perturbation δ such that the controller wins the game $\mathcal{G}_\delta(\mathcal{A})$ when following the lasso $\rho_1\rho_2$ as a strategy. As for the symbolic checking of emptiness of a Büchi timed language [17], we will use a double forward analysis to exhaust all possible lassos, each being tested for robustness by the techniques studied in previous section: a first forward analysis will search for ρ_1, a path from the initial location to an accepting location, and a second forward analysis from each accepting location ℓ to find the cycle ρ_2

around ℓ. Forward analysis means that we compute the successor zone $\mathsf{Post}_\rho(Z)$ when following path ρ from zone Z.

Abstractions of lassos. Before studying in more details the two independant forward analyses, we first study what information we must keep about ρ_1 and ρ_2 in order to still being able to test the robustness of the lasso $\rho_1\rho_2$. A classical problem for robustness is the firing of a *punctual transition*, i.e. a transition where controller has a single choice of time delay: clearly such a firing will be robust for no possible choice of parameter δ. Therefore, we must at least forbid such punctual transitions in our forward analyses. We thus introduce a non-punctual successor operator $\mathsf{Post}^{\mathrm{np}}_\rho$. It consists of the standard successor operator Post_ρ in the timed automaton $\mathcal{A}^{\mathrm{np}}$ obtained from $\mathcal{A}$ by making strict every constraint appearing in the guards ($1 \leq x \leq 2$ becomes $1 < x < 2$). The crucial point is that if a positive delay d can be taken by the controller while satisfying a set of strict constraints, then other delays are also possible, close enough to d. By analogy, a region is said to be *non-punctual* if it contains two valuations separated by a positive time delay. In particular, if such a region satisfies a constraint in $\mathcal{A}$ it also satisfies the corresponding strict constraint in $\mathcal{A}^{\mathrm{np}}$. Therefore, controller wins $\mathcal{G}_\delta(\mathcal{A})$ for some $\delta > 0$ if and only if he wins $\mathcal{G}_\delta(\mathcal{A}^{\mathrm{np}})$ for some $\delta > 0$.

The link between non-punctuality and robustness is as follows:

Theorem 2. *Let $\rho_1\rho_2$ be a lasso of the timed automaton. We have*

$$\exists \delta > 0 \;\; \mathbf{0} \in \mathsf{CPre}^\delta_{\rho_1}(\nu X \;\mathsf{CPre}^\delta_{\rho_2}(X)) \iff \mathsf{Post}^{\mathrm{np}}_{\rho_1}(\mathbf{0}) \cap (\textstyle\bigcup_{\delta>0} \nu X \;\mathsf{CPre}^\delta_{\rho_2}(X)) \neq \emptyset$$

Proof. The proof of this theorem relies on three main ingredients:

1. the timed automaton $\mathcal{A}^{\mathrm{np}}$ allows one to compute $\bigcup_{\delta>0} \mathsf{CPre}^\delta_e(Z')$ by classical predecessor operator: $\mathsf{Pre}^{\mathrm{np}}_e(Z') = \bigcup_{\delta>0} \mathsf{CPre}^\delta_e(Z')$;
2. for all edges e, and zones Z and Z', $Z \cap \mathsf{Pre}^{\mathrm{np}}_e(Z') \neq \emptyset$ if and only if $\mathsf{Post}^{\mathrm{np}}_e(Z) \cap Z' \neq \emptyset$: this duality property on predecessor and successor relations always holds, in particular in $\mathcal{A}^{\mathrm{np}}$. These two ingredients already imply that the theorem holds for a path reduced to a single edge e;
3. we then prove the theorem by induction on length of the path using that $\bigcup_{\delta>0} \mathsf{CPre}^\delta_{\rho_1\rho_2}(Z) = \bigcup_{\delta>0} \mathsf{CPre}^\delta_{\rho_1}\left(\bigcup_{\delta'>0} \mathsf{CPre}^{\delta'}_{\rho_2}(Z)\right)$, due to the monotonicity of the $\mathsf{CPre}^\delta_{\rho_1}$ operator. □

Therefore, in order to test the robustness of the lasso $\rho_1\rho_2$, it is enough to only keep in memory the sets $\mathsf{Post}^{\mathrm{np}}_{\rho_1}(\mathbf{0})$ and $\bigcup_{\delta>0} \nu X \;\mathsf{CPre}^\delta_{\rho_2}(X)$.

Non-punctual forward analysis. As a consequence of the previous theorem, we can use a classical forward analysis of the timed automaton $\mathcal{A}^{\mathrm{np}}$ to look for the prefix ρ_1 of the lasso $\rho_1\rho_2$. A classical inclusion check on zones allows to stop the exploration, this criterion being complete thanks to Theorem 2. It is worth reminding that we consider only bounded clocks, hence the number of reachable zones is finite, ensuring termination.

Robust cycle search. We now perform a second forward analysis, from each possible final location, to find a robust cycle around it. To this end, for each

cycle ρ_2, we must compute the zone $\bigcup_{\delta>0} \nu X\ \mathsf{CPre}^{\delta}_{\rho_2}(X)$. This computation is obtained by arguments developed in Section 4 (Proposition 4). To enumerate cycles ρ_2, we can again use a classical forward exploration, starting from the universal zone $\top$. Using zone inclusion to stop the exploration is not complete: considering a path ρ_2' reaching a zone Z_2' included in the zone Z_2 reachable using some ρ_2, ρ_2' could be robustly iterable while ρ_2 is not. In order to ensure termination of our analysis, we instead use reachability relations inclusion checks. These tests are performed using the technique developed in Section 3, based on constraint graphs (Theorem 1). The correction of this inclusion check is stated in the following lemma, where $\mathsf{Reach}^{\mathrm{np}}_{\rho}$ denotes the reachability relation associated with ρ in the automaton $\mathcal{A}^{\mathrm{np}}$. This result is derived from the analysis based on regions in [25]. Indeed, we can prove that the non-punctual reachability relation we consider captures the existence of non-punctual aperiodic paths in the region automaton, as considered in [25].

Lemma 3. *Let ρ_1 a path from ℓ_0 to some target location ℓ_t. Let ρ_2, ρ_2' be two paths from ℓ_t to some location ℓ, such that $\mathsf{Reach}^{\mathrm{np}}_{\rho_2} \subseteq \mathsf{Reach}^{\mathrm{np}}_{\rho_2'}$. For all paths ρ_3 from ℓ to ℓ_t, $\mathsf{Post}^{\mathrm{np}}_{\rho_1}(\mathbf{0}) \cap (\bigcup_{\delta>0} \nu X\ \mathsf{CPre}^{\delta}_{\rho_2\rho_3}(X)) \neq \emptyset$ implies $\mathsf{Post}^{\mathrm{np}}_{\rho_1}(\mathbf{0}) \cap (\bigcup_{\delta>0} \nu X\ \mathsf{CPre}^{\delta}_{\rho_2'\rho_3}(X)) \neq \emptyset$.*

6 Case study

We implemented our algorithm in C++. To illustrate our approach, we present a case study on the regulation of train networks. Urban train networks in big cities are often particularly busy during rush hours: trains run in high frequency so even small delays due to incidents or passenger misbehavior can perturb the traffic and end up causing large delays. Train companies thus apply regulation techniques: they slow down or accelerate trains, and modify waiting times in order to make sure that the traffic is fluid along the network. Computing robust schedules with provable guarantees is a difficult problem (see e.g. [9]).

We study here a simplified model of a train network and aim at automatically synthesizing a controller that regulates the network despite perturbations, in order to ensure performance measures on total travel time for each train. Consider a circular train network with m stations $s_0, \ldots, s_{m-1}$ and n trains. We require that all trains are at distinct stations at all times. There is an interval of delays $[\ell_i, u_i]$ attached to each station which bounds the travel time from s_i to $s_{i+1 \bmod m}$. Here the lower bound comes from physical limits (maximal allowed speed, and travel distance) while the upper bound comes from operator specification (e.g. it is not desirable for a train to remain at station for more than 3 minutes). The objective of each train i is to cycle on the network while completing each tour within a given time interval $[t_1^i, t_2^i]$.

All timing requirements are naturally encoded with clocks. Given a model, we solve the robust controller synthesis problem in order to find a controller choosing travel times for all trains ensuring a Büchi condition (visiting s_1 infinitely often). Given the fact that trains cannot be at the same station at any given time, it

suffices to state the Büchi condition only for one train, since its satisfaction of the condition necessarily implies that of all other trains.

Let us present two representative instances and then comment the performance of the algorithm on a set of instances. Consider a network with two trains and m stations, with $[\ell_i, u_i] = [200, 400]$ for each station i, and the objective of both trains is the interval $[250 \cdot m, 350 \cdot m]$, that is, an average travel time between stations that lies in $[250, 350]$. The algorithm finds an accepting lasso: intuitively, by choosing δ small enough so that $m\delta < 50$, perturbations do not accumulate too much and the controller can always choose delays for both trains and satisfy the constraints. This case corresponds to scenario A in Figure 4. Consider now the same network but with two different objectives: $[0, 300 \cdot m]$ and $[300 \cdot m, \infty)$. Thus, one train needs to complete each cycle in at most $300 \cdot m$ time units, while the other one in at least $300 \cdot m$ time units. A classical Büchi emptiness check reveals the existence of an accepting lasso: it suffices to move each train in exactly 300 time units between each station. This controller can even recover from perturbations for a bounded number of cycles: for instance, if a train arrives late at a station, the next travel time can be chosen smaller than 300. However, such corrections will cause the distance between the two trains to decrease and if such perturbations happen regularly, the system will eventually enter a deadlock. Our algorithm detects that there is no robust controller for the Büchi objective. This corresponds to the scenario B in Figure 4.

Scenario	m	n	#Clocks	robust?	time
A	6	2	4	yes	4s
B	6	2	4	no	2s
C	6	3	5	no	263s
D	6	3	4	yes	125s
E	6	4	2	yes	53s
F	6	4	2	yes	424s
G	6	4	8		TO
H	6	4	8		TO
I	20	2	2	yes	76s
J	20	2	2	yes	55s
K	30	2	2	yes	579s

Figure 4. Summary of experiments with different sizes. In each scenario, we assign a different objective to a subset of trains. The answer is *yes* if if a robust controller was found, *no* if none exists. TO stands for a time-out of 30 minutes.

Figure 4 summarizes the outcome of our prototype implementation on other scenarios. The tool was run on a 3.2Ghz Intel i7 processor running Linux, with a 30 minute time out and 2GB of memory. The performance is sensitive to the number of clocks: on scenarios with 8 clocks the algorithm ran out of time.

7 Conclusion

Our case study illustrates the application of robust controller synthesis in small or moderate size problems. Our prototype relies on the DBM libraries that we use with twice as many clocks to store the constraints of the normalised constraint graphs. In order to scale to larger models, we plan to study extrapolation operators and their integration in the computation of reachability relations, which seems to be a challenging task. Different strategies can also be adopted for the double forward analysis, switching between the two modes using heuristics, a parallel implementation, etc.

References

1. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. Giovanni Bacci, Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Pierre-Alain Reynier. Optimal and robust controller synthesis using energy timed automata with uncertainty. In Bill W. Roscoe and Jan Peleska, editors, *Proceedings of the 22nd International Symposium on Formal Methods (FM'18)*, Lecture Notes in Computer Science, pages 203–221, Oxford, UK, July 2018. Springer. Best paper award.
3. Johan Bengtsson and Wang Yi. *Timed Automata: Semantics, Algorithms and Tools*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2004.
4. Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In R. E. A. Mason, editor, *Information Processing 83 – Proceedings of the 9th IFIP World Computer Congress (WCC'83)*, pages 41–46. North-Holland/IFIP, September 1983.
5. Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 2005*, volume 3653, pages 66–80. Springer-Verlag, 2005.
6. Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In Claire Tomlin and Mark R. Greenstreet, editors, *Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control (HSCC'02)*, volume 2289 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2002.
7. Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Timed parity games: Complexity and robustness. *Logical Methods in Computer Science*, 7(4), 2011.
8. Hubert Comon-Lundh and Yan Jurski. Timed automata and the theory of real numbers. In *Proceedings of CONCUR'99*, volume 1664 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 1999.
9. A. D'Ariano, M. Pranzo, and I. A. Hansen. Conflict resolution and train speed coordination for solving real-time timetable perturbations. *IEEE Transactions on Intelligent Transportation Systems*, 8(2):208–222, June 2007.
10. David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems (CAV 1989)*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1990.
11. Thomas A. Henzinger, Jan Otop, and Roopsha Samanta. Lipschitz robustness of timed i/o systems. In Barbara Jobstmann and K. Rustan M. Leino, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 250–267, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
12. Thomas A. Henzinger and Joseph Sifakis. The embedded systems design challenge. In *FM 2006*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.
13. Frédéric Herbreteau and B. Srivathsan. Efficient on-the-fly emptiness check for timed büchi automata. In *ATVA 2010*, volume 6252 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2010.
14. Frédéric Herbreteau, B. Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz. Why liveness for timed automata is hard, and what we can do about it. In *FSTTCS 2016*, volume 65 of *LIPIcs*, pages 48:1–48:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

15. Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Efficient emptiness check for timed büchi automata. *Formal Methods in System Design*, 40(2):122–146, 2012.
16. Rémi Jaubert and Pierre-Alain Reynier. Quantitative robustness analysis of flat timed automata. In *Proceedings of the 14th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'11)*, volume 6604 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2011.
17. Alfons Laarman, Mads Chr. Olesen, Andreas Engelbredt Dalsgaard, Kim Guldstrand Larsen, and Jaco van de Pol. Multi-core emptiness checking of timed büchi automata using inclusion abstraction. In *CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 968–983. Springer, 2013.
18. Kim G. Larsen, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. Robust synthesis for real-time systems. *Theor. Comput. Sci.*, 515:96–122, 2014.
19. Guangyuan Li. Checking timed büchi automata emptiness using lu-abstractions. In *FORMATS 2009*, volume 5813 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2009.
20. Pavithra Prabhakar and Miriam García Soto. Formal synthesis of stabilizing controllers for switched systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, HSCC '17, pages 111–120, New York, NY, USA, 2017. ACM.
21. Pavithra Prabhakar and Miriam Garcia Soto. Counterexample guided abstraction refinement for stability analysis. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, pages 495–512, 2016.
22. Karin Quaas, Mahsa Shirmohammadi, and James Worrell. Revisiting reachability in timed automata. In *LICS'17*. IEEE, 2017.
23. Nima Roohi, Pavithra Prabhakar, and Mahesh Viswanathan. Robust model checking of timed automata under clock drifts. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, HSCC '17, pages 153–162, New York, NY, USA, 2017. ACM.
24. Ocan Sankur, Patricia Bouyer, and Nicolas Markey. Shrinking Timed Automata. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *LIPIcs*, pages 90–102. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011.
25. Ocan Sankur, Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust controller synthesis in timed automata. In *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR'13)*, volume 8052 of *Lecture Notes in Computer Science*, pages 546–560. Springer, 2013.
26. Thanh-Tung Tran. *Verification of timed automata : reachability, liveness and modelling. (Vérification d'automates temporisés : sûreté, vivacité et modélisation)*. PhD thesis, University of Bordeaux, France, 2016.
27. Stavros Tripakis. Checking timed büchi automata emptiness on simulation graphs. *ACM Trans. Comput. Log.*, 10(3):15:1–15:19, 2009.
28. Stavros Tripakis, Sergio Yovine, and Ahmed Bouajjani. Checking timed büchi automata emptiness efficiently. *Formal Methods in System Design*, 26(3):267–292, 2005.