Robust Reachability in Timed Automata: A Game-based Approach *

Patricia Bouyer, Nicolas Markey, and Ocan Sankur

LSV, CNRS & ENS Cachan, France.
{bouyer,markey,sankur}@lsv.ens-cachan.fr

Abstract. Reachability checking is one of the most basic problems in verification. By solving this problem, one synthesizes a strategy that dictates the actions to be performed for ensuring that the target location is reached. In this work, we are interested in synthesizing "robust" strategies for ensuring reachability of a location in a timed automaton; with "robust", we mean that it must still ensure reachability even when the delays are perturbed by the environment. We model this perturbed semantics as a game between the controller and its environment, and solve the parameterized robust reachability problem: we show that the existence of an upper bound on the perturbations under which there is a strategy reaching a target location is EXPTIME-complete.

1 Introduction

Timed automata [2] are a timed extension of finite-state automata. They come with an automata-theoretic framework to design, model, verify and synthesize systems with timing constraints. One of the most basic problems in timed automata is the reachability problem: given a timed automaton and a target location, is there a path that leads to that location? This can be rephrased in the context of control as follows: is there a *strategy* that dictates how to choose time delays and edges to be taken so that a target location is reached? This problem has been solved long ago [2], and efficient algorithms have then been developed and implemented [13, 18].

However, the abstract model of timed automata is an idealization of real timed systems. For instance, we assume in timed automata that strategies can choose the delays with arbitrary precision. In particular, the delays can be arbitrarily close to zero (the system is arbitrarily fast), and clock constraints can enforce exact delays (time can be measured exactly). Although these assumptions are natural in abstract models, they need to be justified after the design phase. Indeed the situation is different in real-world systems: digital systems have response times that may not be negligible, and control software cannot ensure timing constraints exactly, but only up to some error, caused by clock imprecisions, measurement errors, and communication delays. A good control software must be *robust*, *i.e.*, it must ensure good behavior in spite of small imprecisions [11, 12].

* This work has been partly supported by project ImpRo (ANR-10-BLAN-0317)

In this work, we are interested in the synthesis of robust strategies in timed automata for reachability objectives, taking into account response times and imprecisions. We propose to model the problem as a game between a controller (that will guide the system) and its environment. In our semantics, which is parameterized by some $0 < \delta_P \leq \delta_R$, the controller chooses to delay an amount $d \geq \delta_R$, and the system delays d', where d' is chosen by the environment satisfying $|d - d'| \leq \delta_P$. We say that a given location is *robustly reachable* if there exist parameters $0 < \delta_P \leq \delta_R$ such that the controller has a winning strategy ensuring that the location is reached against any strategy of the environment. If δ_P and δ_R are fixed, this can be solved using techniques from control theory [3]. However δ_P, δ_R are better seen as parameters here, representing imprecisions in the implementation of the system (they may depend on the digital platform on which the system is implemented), and whose values may not be available in the design phase. To simplify the presentation, but w.l.o.g., we assume in this paper that $\delta = \delta_P = \delta_R$; our algorithm can easily be adapted to the general case (by adapting the shrink operator in Section 3).

Note that this semantics was studied in [6] for timed games with *fixed* parameters, where the parameterized version was presented as a challenging open problem. We solve this problem for reachability objectives in timed automata: we show that deciding the existence of $\delta > 0$, and of a strategy for the controller so as to ensure reachability of a given location (whatever the imprecision, up to δ), is EXPTIME-complete. Moreover, if there is a strategy, we can compute a *uniform* one, which is parameterized by δ, using *shrunk difference bound matrices* (shrunk DBMs) that we introduced recently [17]. In this case, our algorithm provides a bound $\delta_0 > 0$ such that the strategy is correct for all $\delta \in [0, \delta_0]$. Our strategies also give quantitative information on how perturbations accumulate or can compensate. Technically, our work extends shrunk DBMs by *constraints*, and establishes non-trivial algebraic properties of this data structure (Section 3). The main result is then obtained by transforming the infinite-state game into a finite abstraction, which we prove can be used to symbolically compute a winning strategy, if any (see Section 4).

By lack of space, technical proofs have been omitted; they can be found in [5].

2 Robust reachability in timed automata

2.1 Timed automata and robust reachability

Given a finite set of clocks $\mathcal{C}$, we call *valuations* the elements of $\mathbb{R}^{\mathcal{C}}_{\geq 0}$. For a subset $R \subseteq \mathcal{C}$ and a valuation v, $v[R \leftarrow 0]$ is the valuation defined by $v[R \leftarrow 0](x) = v(x)$ for $x \in \mathcal{C} \setminus R$ and $v[R \leftarrow 0](x) = 0$ for $x \in R$. Given $d \in \mathbb{R}_{\geq 0}$ and a valuation v, the valuation $v + d$ is defined by $(v + d)(x) = v(x) + d$ for all $x \in \mathcal{C}$. We extend these operations to sets of valuations in the obvious way. We write $\mathbf{0}$ for the valuation that assigns 0 to every clock.

An atomic clock constraint is a formula of the form $k \preceq x \preceq' l$ or $k \preceq x - y \preceq' l$ where $x, y \in \mathcal{C}$, $k, l \in \mathbb{Z} \cup \{-\infty, \infty\}$ and $\preceq, \preceq' \in \{<, \leq\}$. A *guard* is a conjunction of atomic clock constraints. A valuation v satisfies a guard g, denoted $v \models g$, if all constraints are satisfied when each $x \in \mathcal{C}$ is replaced with $v(x)$.

Definition 1 ([2]). *A* timed automaton $\mathcal{A}$ *is a tuple* $(\mathcal{L}, \mathcal{C}, \ell_0, E)$*, consisting of finite sets* $\mathcal{L}$ *of locations,* $\mathcal{C}$ *of clocks,* $E \subseteq \mathcal{L} \times \Phi_{\mathcal{C}} \times 2^{\mathcal{C}} \times \mathcal{L}$ *of edges, and where* $\ell_0 \in \mathcal{L}$ *is the initial location. An edge* $e = (\ell, g, R, \ell')$ *is also written as* $\ell \xrightarrow{g,R} \ell'$*.*

Standard semantics of timed automata is usually given as a timed transition system. To capture robustness, we define the semantics as a game where perturbations in delays are uncontrollable. Given a timed automaton $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, E)$ and $\delta > 0$, we define the *perturbation game* of $\mathcal{A}$ w.r.t. δ as a two-player turn-based timed game $\mathcal{G}_\delta(\mathcal{A})$ between players *Controller* and *Perturbator*. The state space of $\mathcal{G}_\delta(\mathcal{A})$ is partitioned into $V_C \cup V_P$ where $V_C = \mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}$ is the set of states that belong to Controller and $V_P = \mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0} \times \mathbb{R}_{\geq 0} \times E$ is the set of states that belong to Perturbator. The initial state is $(\ell_0, \mathbf{0})$ and belongs to Controller. The transitions are defined as follows: from any state $(\ell, v) \in V_C$, there is a transition to $(\ell, v, d, e) \in V_P$ whenever $d \geq \delta$, $e = (\ell, g, R, \ell')$ is an edge such that $v + d \models g$. Then, from any such state $(\ell, v, d, e) \in V_P$, there is a transition to $(\ell', (v + d + \epsilon)[R \leftarrow 0]) \in V_C$, for any $\epsilon \in [-\delta, \delta]$.

We assume familiarity with basic notions in game theory, and quickly survey the main definitions. A run in $\mathcal{G}_\delta(\mathcal{A})$ is a finite or infinite sequence of consecutive states starting at $(\ell_0, \mathbf{0})$. It is said maximal if it is infinite or cannot be extended. A strategy for Controller is a function that assigns to every non-maximal run ending in some $(\ell, v) \in V_C$, a pair (d, e) where $d \geq \delta$ and e is an edge enabled at $v + d$ (i.e., there is a transition from (ℓ, v) to (ℓ, v, d, e)). A run ρ is compatible with a strategy f if for every prefix ρ' of ρ ending in V_C, the next transition along ρ after ρ' is given by f. Given a target location ℓ, a strategy f is winning for the reachability objective defined by ℓ whenever all maximal runs that are compatible with f visit ℓ.

Observe that we require at any state (ℓ, v), that Controller should choose a delay $d \geq \delta$ and an edge e that is enabled after the chosen delay d. The edge chosen by Controller is always taken but there is no guarantee that the guard will be satisfied exactly when the transition takes place. In fact, Perturbator can perturb the delay d chosen by Controller by any amount $\epsilon \in [-\delta, \delta]$, including those that do not satisfy the guard. Notice that $\mathcal{G}_0(\mathcal{A})$ corresponds to the standard (non-robust) semantics of $\mathcal{A}$. We are interested in the following problem.

Problem 1 (Parameterized Robust Reachability). Given a timed automaton $\mathcal{A}$ and a target location ℓ, decide whether there exists $\delta > 0$ such that Controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$ for the reachability objective ℓ.

Notice that we are interested in the parameterized problem: δ is not fixed in advance. For fixed parameter, the problem can be formulated as a usual timed game, see [6]. Our main result is the decidability of this parameterized problem. Moreover, if there is a solution, we compute a strategy represented by parameterized difference-bound matrices where δ is the parameter; the strategy is thus *uniform* with respect to δ. In fact, we provide a bound $\delta_0 > 0$ such that the strategy is winning for Controller for *any* $\delta \in [0, \delta_0]$. These strategies also provide a quantitative information on how much the perturbation accumulates (See Fig. 3). The main result of this paper is the following:

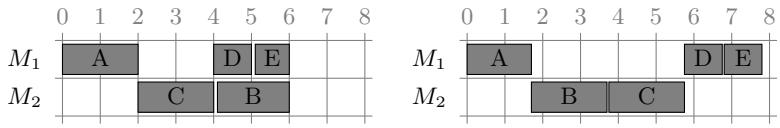
Theorem 2. *Parameterized robust reachability is* EXPTIME*-complete.*

Checking parameterized robust reachability is different from usual reachability checking mainly for two reasons. First, in order to reach a given location, Controller has to choose the delays along a run, so that these perturbations do not accumulate and block the run. In particular, it shouldn't play too close to the borders of the guards (see Fig. 3). Second, due to these uncontrollable perturbations, some regions that are not reachable in the absence of perturbation can become reachable (see Fig. 4). So, Controller must also be able to win from these new regions. The regions that become reachable in our semantics are those *neighboring* reachable regions. The characterization of these neighboring regions is one of the main difficulties in this paper (see Section 3.5).

2.2 Motivating example: robust real-time scheduling

An application of timed automata is the synthesis of schedulers in various contexts [1]. We show that robust reachability can help providing a better schedulability analysis: we show that schedulers synthesized by standard reachability analysis may not be robust: even the slightest *decrease* in task execution times can result in a large *increase* in the total time. This is a phenomenon known as *timing anomalies*, first identified in [9].

Consider the scheduling problem described in Fig. 1, inspired by [16]. Assume that we look for a *greedy* (*i.e.*, work-conserving) scheduler, that will immediately start executing a task if a machine is free for execution on an available task. What execution time can guarantee a greedy scheduling policy on this instance? One can model this problem as a timed automaton, and prove, by classical reachability analysis, that these tasks can be scheduled using a greedy policy within six time units. However the scheduler obtained this way may not be robust, as illustrated in Fig. 1(b). If the duration of task A unexpectedly drops by a small amount $\delta > 0$, then any greedy scheduler will schedule task B before task C, since the latter is not ready for execution at time $2 - \delta$. This yields a scheduling of tasks in $8 - \delta$ time units.



(a) A has duration 2. (b) A has duration $2 - \delta$.

Fig. 1. Consider tasks A, B, C of duration 2 and D, E of duration 1. Dependences between tasks are as follows: $A \to B$ and $C \to D, E$, meaning *e.g.* that A must be completed before B can start. Task A must be executed on machine M_1 and tasks B, C on machine M_2. Moreover, task C cannot be scheduled before 2 time units (which could be modelled using an extra task). Fig. 1(a) shows the optimal greedy schedule for these tasks under these constraints, while Fig. 1(b) shows the outcome of any greedy scheduler when the duration of task A is less than 2.

Our robust reachability algorithm is able to capture such phenomena, and can provide correct and robust schedulers. In fact, it would answer that the tasks are not schedulable in six time units (with a greedy policy), but only in eight time units.

2.3 Related work: robustness in timed automata and games

There has been a recent effort to consider imprecisions inherent to real systems in the theory of timed systems. In particular there has been several attempts to define convenient notions of robustness for timed automata, see [14] for a survey.

The approach initiated in [15, 8, 7] is the closest to our framework/proposition. It consists in *enlarging* all clocks constraints of the automaton by some parameter δ, that is transforming each constraint of the form $x \in [a, b]$ into $x \in [a-\delta, b+\delta]$, and in synthesizing $\delta > 0$ such that all runs of the enlarged automaton satisfy a given property. This can be reformulated as follows: does there exists some $\delta > 0$ such that whatever Controller and Perturbator do in $\mathcal{G}_\delta(\mathcal{A})$, a given property is satisfied. This is therefore the universal counterpart of our formulation of the parameterized robustness problem. It has been shown that this universal parameterized robust model-checking is no more difficult (in terms of complexity) than standard model-checking. This has to be compared with our result, where complexity goes up from PSPACE to EXPTIME.

Another work that is close to ours is that of [6]. The authors consider general two-player (concurrent) games with a fixed lower bound on delays, where chosen delays can be changed by some fixed value δ. It is then shown that winning strategies can be synthesized: In fact, when δ is fixed, the semantics can simply be encoded by a usual timed game, and standard algorithms can be applied. Whether one can synthesize $\delta > 0$ for which the controller has a winning strategy was left as a challenging open problem. We partially solve this open problem here, under the assumption that there is a single player with a reachability objective. The extension to two-player games (with reachability objective) is ongoing work, and we believe the techniques presented in this paper can be used for that purpose.

Finally, [10] studies a topological and language-based approach to robustness, where (roughly) a timed word is accepted by the automaton if, and only if, one of its neighborhoods is accepted. This is not related to our formalization.

3 Shrinking DBMs

3.1 Regions, zones and DBMs

We assume familiarity with the notions of regions and zones (see [4]). For two regions r and r', we write $r \lessdot r'$ if r' is the immediate (strict) time-successor of r. A *zone* is a set of clock valuations satisfying a guard.

We write $\mathcal{C}_0$ for the set $\mathcal{C} \cup \{0\}$. A *difference-bound matrix (DBM)* is a $|\mathcal{C}_0| \times |\mathcal{C}_0|$-matrix over $(\mathbb{R} \times \{<, \leq\}) \cup \{(\infty, <)\}$. A DBM M naturally represents a zone (which we abusively write M as well), defined as the set of valuations v

such that, for all $x, y \in \mathcal{C}_0$, writing $(M_{x,y}, \prec_{x,y})$ for the (x,y)-entry of M, it holds $v(x) - v(y) \prec_{x,y} M_{x,y}$ (where $v(0) = 0$). For any DBM M, let $\mathsf{G}(M)$ denote the graph over nodes $\mathcal{C}_0$, where the weight of the edge $(x,y) \in \mathcal{C}_0^2$ is $(M_{x,y}, \prec_{x,y})$. The normalization of M corresponds to assigning to each edge (x,y) the weight of the shortest path in $\mathsf{G}(M)$. We say that M is *normalized* when it is stable under normalization.

3.2 Shrinking

Consider the automaton $\mathcal{A}$ of Fig. 2, where the goal is to reach ℓ_3. If there is no perturbation or lower bound on the delays between transitions (*i.e.*, $\delta = 0$), then the states from which Controller can reach location ℓ_3 can be computed backwards. One can reach ℓ_3 from location ℓ_2 and any state in the zone $X = (x \leq 2) \wedge (y \leq 1) \wedge (1 \leq x - y)$, shown by (the union of the light and dark) gray areas on Fig. 3 (left); this is the set of time-predecessors of the corresponding guard. The set of winning states from location ℓ_1 is the zone $Y = (x \leq 2)$, shown in Fig. 3 (right), which is simply the set of predecessors of X at ℓ_2. When $\delta > 0$ however, the set of winning states at ℓ_2 is a "shrinking" of X, shown by the dark gray area. If the value of the clock x is too close to 2 upon arrival in ℓ_2, Controller will fail to satisfy the guard $x = 2$ due to the lower bound δ on the delays. Thus, the winning states from ℓ_2 are described by $X \cap (x \leq 2 - \delta)$. Then, this shrinking is backward propagated to ℓ_1: the winning states are $Y \cap (x \leq 2 - 2\delta)$, where we "shrink" Y by 2δ in order to compensate for a possible perturbation.

An important observation here is that when $\delta > 0$ is small enough, so that both $X \cap (x \leq 2 - \delta)$ and $Y \cap (x \leq 2 - 2\delta)$ are non-empty, these sets precisely describe the winning states. Thus, we have a *uniform* description of the winning states for "all small enough $\delta > 0$". We now define *shrunk DBMs*, a data structure we introduced in [17], in order to manipulate "shrinkings" of zones.

3.3 Shrunk DBMs

For any interval $[a,b]$, we define the *shrinking operator* as $\mathsf{shrink}_{[a,b]}(Z) = \{v \mid v + [a,b] \subseteq Z\}$ for any zone Z. We only use operators $\mathsf{shrink}_{[0,\delta]}$ and $\mathsf{shrink}_{[-\delta,\delta]}$ in the sequel. For a zone Z represented as a DBM, $\mathsf{shrink}_{[0,\delta]}(Z)$ is the DBM $Z - \delta \cdot \mathbb{1}_{\mathcal{C}\times\{0\}}$ and $\mathsf{shrink}_{[-\delta,\delta]}(Z)$ is the DBM $Z - \delta \cdot \mathbb{1}_{\mathcal{C}\times\{0\}\cup\{0\}\times\mathcal{C}}$, for any $\delta > 0$.

Our aim is to handle these DBMs symbolically. For this, we define *shrinking matrices (SM)*, which are nonnegative integer square matrices with zeroes on their diagonals. A *shrunk DBM* is then a pair (M, P) where M is a DBM, P is a

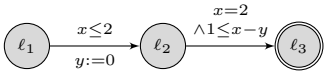


Fig. 2. Automaton $\mathcal{A}$

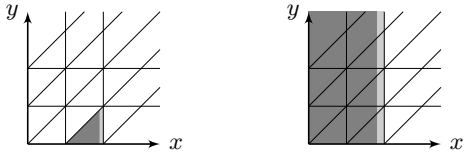


Fig. 3. Winning states in ℓ_2 (left) and in ℓ_1 (right)

shrinking matrix [17]. The meaning of this pair is that we consider DBMs $M-\delta P$ where $\delta \in [0, \delta_0]$ for some $\delta_0 > 0$. In the sequel, we abusively use "for all small enough $\delta > 0$" meaning "there exists $\delta_0 > 0$ such that for all $\delta \in [0, \delta_0]$". We also adopt the following notation: when we write a statement involving a shrunk DBM (M, P), we mean that the statement holds for $(M-\delta P)$ for all small enough $\delta > 0$. For instance, $(M, P) = \mathsf{Pre}_{\mathsf{time}}((N, Q))$ means that $M - \delta P = \mathsf{Pre}_{\mathsf{time}}((N - \delta Q))$ for all small enough $\delta > 0$. In the same vein, shrunk DBMs can be re-shrunk, and we write $\mathsf{shrink}((M, P))$ (resp. $\mathsf{shrink}^+((M, P))$) for the shrunk DBM (N, Q) such that $N - \delta Q = \mathsf{shrink}_{[-\delta,\delta]}(M - \delta P)$ (resp. $N - \delta Q = \mathsf{shrink}_{[0,\delta]}(M - \delta P)$) for all small enough $\delta > 0$.

It was shown in [17] that when usual operations are applied on shrunk DBMs, one always obtain shrunk DBMs, whose shrinking matrices can be computed. We refer to [4, 17] for the formal definitions of these operations.

Lemma 3 ([17]). *Let $M = f(N_1, \ldots, N_k)$ be an equation between normalized DBMs $M, N_1, \ldots, N_k$, using the operators Pre_{time}, $\mathsf{Unreset}_R$, $\cap$, shrink and shrink^+ and let $P_1, \ldots, P_k$ be SMs. Then, there exists a SM Q such that (M, Q) is normalized and $(M, Q) = f\big((N_1, P_1), \ldots, (N_k, P_k)\big)$. Moreover, Q and the corresponding upper bound on δ can be computed in polynomial time.*

3.4 Shrinking constraints

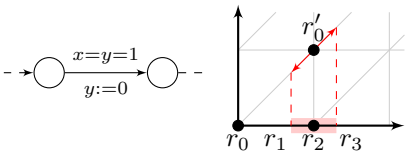
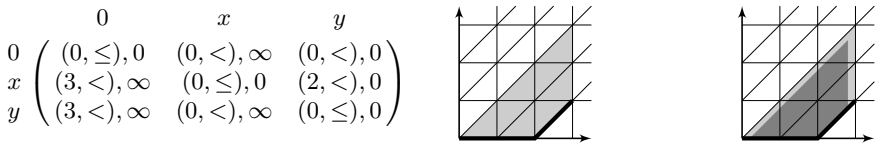


Fig. 4. Perturbing one transition

Consider a transition of a timed automaton, as depicted on the figure at right. From region r_0, the game can reach regions r_1, r_2, r_3, depending on the move of Perturbator. Therefore, in order to win, Controller needs a winning strategy from all three regions. One can then inductively look for winning strategies from these regions; this will generally require shrinking, as exemplified in Fig. 3. However, not all shrinkings of these regions provide a winning strategy from r_0. In fact, r_1 (resp. r_3) should not shrink from the right (resp. left) side: their union should include the shaded area, thus points that are arbitrarily close to r_2. In order to define the shrinkings that are useful to us, we introduce shrinking constraints.

Definition 4. *Let M be a DBM. A* shrinking constraint *for M is a $|\mathcal{C}_0| \times |\mathcal{C}_0|$ matrix over $\{0, \infty\}$. A shrinking matrix P is said to* respect *a shrinking constraint S if $P \leq S$, where the comparison is component-wise. A pair $\langle M, S\rangle$ of a DBM and a shrinking constraint is called a* constrained DBM.

Shrinking constraints specify which facets of a given zone one is (not) allowed to shrink (see Fig. 5). A shrinking constraint S for a DBM M is said to be *well* if for any SM $P \leq S$, (M, P) is non-empty. A *well constrained DBM* is a constrained DBM given with a well shrinking constraint. We say that a shrinking constraint S for a DBM M is *normalized* if it is the minimum among all equivalent shrinking constraints: for any shrinking constraint S' if for all SMs



(a) A constrained DBM $\langle M, S\rangle$ and its representation (b) A shrinking of $\langle M, S\rangle$

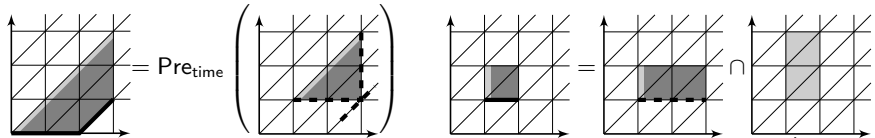
Fig. 5. Consider a zone defined by $0 < x < 3$, $0 < y < 3$, and $0 < x - y < 2$. Let the shrinking constraint S be defined by $S_{0,y} = 0$, $S_{x,y} = 0$, and $S_{z,z'} = \infty$ for other components. The resulting $\langle M, S\rangle$ is depicted on the left, as a matrix (where, for convenience, we merged both matrices into a single one) and as a constrained zone (where a thick segment is drawn for any boundary that is not "shrinkable", i.e., with $S_{z,z'} = 0$). On the right, the dark gray area represents a shrinking of M that satisfies S.

P, $P \leq S \Leftrightarrow P \leq S'$, then $S \leq S'$. One can show that any shrinking constraint can be made normalized, by a procedure similar to the normalization of DBMs. Lemma 5 shows that shrinking constraints can be propagated along operations on DBMs. This is illustrated in Fig. 6.

Lemma 5. *Let M, N, N' be normalized non-empty DBMs.*

1. *Assume that $M = \mathsf{Pre}_{\mathsf{time}}(N)$, $M = N \cap N'$, or $M = \mathsf{Unreset}_R(N)$. Then, for any normalized well shrinking constraint S for M, there exists a well shrinking constraint S' for N such that for any SM Q, the following holds: $Q \leq S'$ iff the SM P s.t. $(M, P) = \mathsf{Pre}_{\mathsf{time}}((N, Q))$ (respectively, $(M, P) = (N, Q) \cap N'$ or $(M, P) = \mathsf{Unreset}_R((N, Q))$) satisfies $P \leq S$.*
2. *Assume that $M = N \cap N'$. For any well shrinking constraint S for N, there exists a shrinking constraint S' for M such that for any SM Q, the following holds: $Q \leq S'$ iff a SM $P \leq S$ s.t. $(N, P) \cap N' \subseteq (M, Q)$. Moreover, if $(N, P) \cap N' \neq \emptyset$ for all SMs $P \leq S$, then S' is well.*

Let us comment on Fig. 6(a), and how it can be used for our purpose. Assume there is an edge guarded by N (the whole gray area in the right) without resets. In the non-robust setting, this guard can be reached from any point of M (the whole gray area in the left). If we have a shrinking constraint S on M, and we



(a) $(M, P) = \mathsf{Pre}_{\mathsf{time}}((N, Q))$ (b) $(M, P) = (N, Q) \cap N'$

Fig. 6. The figures illustrate the first item in Lemma 5. In each case, DBMs M, N and N' are fixed and satisfy the "unshrunk" equation. The thick plain segments represent the fixed shrinking constraint S. The dashed segments represent resulting constraint S'. For any SM Q, we have $Q \leq S'$ iff there is an SM $P \leq S$ that satisfies the equation.

want to synthesize a winning strategy from a shrinking of M satisfying S, then Lemma 5 gives the shrinking constraint S' for N, with the following property: given any shrinking (N, Q), we can find $P \leq S$ with $(M, P) = \mathsf{Pre}_{\mathsf{time}}((N, Q))$ (hence, we can delay into (N, Q)), if, and only if Q satisfies $Q \leq S'$. The problem is now "reduced" to finding a winning strategy from $\langle N, S' \rangle$. However, forward-propagating these shrinking constraints is not always that easy. We also need to deal with resets, with the fact that Controller has to choose a delay greater than $\delta > 0$, and also with the case where there are several edges leaving a location. This is the aim of the following developments.

3.5 Neighborhoods

We now consider constrained regions, which are constrained DBMs in which the DBM represents a region. Fig. 4 shows that if Controller plays to a region, then Perturbator can reach some of the surrounding regions, shown by the arrows. To characterize these, we define the set of *neighboring regions* of $\langle r, S \rangle$ as,

$$\mathcal{N}_{r,S} = \left\{ r' \;\middle|\; r' \lessdot^* r \text{ or } r \lessdot^+ r',\ \text{and } \forall Q \leq S.\ r' \cap \mathsf{enlarge}((r, Q)) \neq \emptyset \right\}$$

where $\mathsf{enlarge}((r, Q))$ is the shrunk DBM (M, P) such that $v + [-\delta, \delta] \subseteq M - \delta P$ for every $v \in r - \delta Q$. This is the set of regions that have "distance" at most δ to any shrinking of the constrained region (r, S). We write $\mathsf{neighbor}\langle r, S \rangle = \bigcup_{r' \in \mathcal{N}_{r,S}} r'$.

Lemma 6 (Neighborhood). *Let $\langle r, S \rangle$ be a well constrained region. Then $\mathsf{neighbor}\langle r, S \rangle$ is a zone. If N is the corresponding normalized DBM, there exists a well shrinking constraint S' such that for every SM Q, $Q \leq S'$ iff the SM P defined by $(r, P) = r \cap \mathsf{shrink}((N, Q))$, satisfies $P \leq S$. The pair $\langle N, S' \rangle$ is the* constrained neighborhood *of $\langle r, S \rangle$, and it can be computed in polynomial time.*

Constrained neighborhoods are illustrated in Fig. 7.

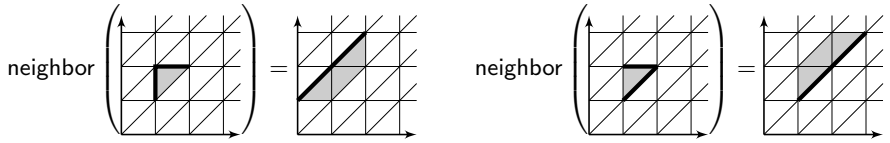


Fig. 7. Constrained neighborhood of two constrained regions. Notice that inside any shrinking of the constrained region, there is always a valuation such that a perturbation of $[-\delta, \delta]$ moves the valuation to any region of the neighborhood.

3.6 Two crucial properties for the construction of the abstraction

The following lemma characterizes, given a constrained region $\langle r, S \rangle$, the set of constrained regions $\langle r', S_{r'} \rangle$ such that any shrunk region satisying $\langle r', S_{r'} \rangle$ can be reached by delaying from some shrunk region satisfying $\langle r, S \rangle$.

Lemma 7. *Let $\langle r, S\rangle$ be a well constrained region, and r' be a region such that $r \lessdot^* r'$. Then the following properties are equivalent:*

1. *there exists a well shrinking constraint S' (which can be computed in polynomial time) such that for every SM Q, $Q \leq S'$ iff the SM P such that $(r, P) = r \cap \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}((r', Q)))$, satisfies $P \leq S$;*
2. *$\mathsf{neighbor}\langle r, S\rangle \subseteq \mathsf{Pre}_{\mathsf{time}}(r')$;*

Note that this lemma may not hold for all r' with $r \lessdot r'$. Consider the constrained region $\langle r, S\rangle$ on the right of Fig. 7, and let r' be the first triangle region above r: any valuation arbitrarily close to the thick segments will be in $r - \delta P$ for any $P \leq S$, but it can only reach r' by delaying less than δ time units.

Lemma 8. *Let $\langle r, S\rangle$ be a well constrained region, and let $R \subseteq \mathcal{C}$. Let $\mathcal{N}$ be the set of neighboring regions of $\langle r, S\rangle$, and $\mathcal{N}' = \{r'[R \leftarrow 0] \mid r' \in \mathcal{N}\}$. Then, there exist well shrinking constraints $S_{r''}$ for all $r'' \in \mathcal{N}'$ such that for any $(Q_{r''})_{r'' \in \mathcal{N}'}$, we have $Q_{r''} \leq S_{r''}$ for all $r'' \in \mathcal{N}'$ iff there exists $P \leq S$ such that*

$$(r, P) \subseteq r \cap \mathsf{shrink}\Big(\bigcup_{r' \in \mathcal{N}} (r' \cap \mathsf{Unreset}_R((r'', Q_{r''})))\Big).$$

with $r'' = r'[R \leftarrow 0]$. Moreover, all $\langle r'', S_{r''}\rangle$ can be computed in polynomial time.

This lemma gives for instance the shrinking constraints that should be satisfied in r_1, r_2 and r_3, in Fig. 4, once shrinking constraint in r'_0 is known. In this case, the constraint in r'_0 is 0 everywhere since it is a punctual region. The neighborhood $\mathcal{N}$ of r'_0 is composed of r'_0 and two extra regions (defined by $(0 < x < 1) \wedge (x = y)$ and $(1 < x < 2) \wedge (x = y)$). If there are shrinkings of regions r_1, r_2, r_3 satisfying the corresponding shrinking constraints (given in the lemma), and from which Controller wins, then one can derive a shrinking of r'_0, satisfying its constraint, and from which Controller wins. In the next section, we define the game $\mathcal{RG}(\mathcal{A})$ following this idea, and explain how it captures the game semantics for robustness.

4 A finite game abstraction

Let $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, E)$ be a timed automaton. We define a finite turn-based game $\mathcal{RG}(\mathcal{A})$ on a graph whose nodes are of two sorts: *square nodes* labelled by (ℓ, r, S_r), where ℓ is a location, r a region, S_r is a well shrinking constraint for r; *diamond nodes* labelled similarly by (ℓ, r, S_r, e) where moreover e is an edge leaving ℓ. Square nodes belong to Controller, while diamond nodes belong to Perturbator. Transitions are defined as follows:

(a) From each square node (ℓ, r, S_r), for any edge $e = (\ell, g, R, \ell')$ of $\mathcal{A}$, there is a transition to the diamond node $(\ell, r', S_{r'}, e)$ if the following conditions hold:
   (i) $r \lessdot^* r'$ and $r' \subseteq g$;
   (ii) $S_{r'}$ is such that for all SMs Q, $Q \leq S_{r'}$ iff there exists $P \leq S_r$ with

$$(r, P) = r \cap \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}((r', Q)))$$

(*b*) From each diamond node (ℓ, r, S_r, e), where $e = (\ell, g, R, \ell')$ is an edge of $\mathcal{A}$, writing $\mathcal{N}$ for the set of regions in the neighborhood of (r, S_r) and $\mathcal{N}' = \{r'[R \leftarrow 0] \mid r' \in \mathcal{N}\}$, there are transitions to all square nodes $(\ell', r'', S_{r''})$ with $r'' \in \mathcal{N}'$, and $(S_{r''})_{r'' \in \mathcal{N}'}$ are such that for all SMs $(Q_{r''})_{r'' \in \mathcal{N}'}$, it holds $Q_{r''} \leq S_{r''}$ for every $r'' \in \mathcal{N}'$ iff there exists $P \leq S_r$ such that

$$(r, P) \subseteq r \cap \mathsf{shrink}(\bigcup_{r' \in \mathcal{N}} (r' \cap \mathsf{Unreset}_R((r'', Q_{r''}))) \qquad (\text{where } r'' = r'[R \leftarrow 0])$$

Intuitively, the transitions from the square nodes are the decisions of Controller. In fact, it has to select a delay and a transition whose guard is satisfied. Then Perturbator can choose any region in the neighborhood of the current region, and, after reset, this determines the next state.

Note that $\mathcal{RG}(\mathcal{A})$ can be computed, thanks to Lemmas 7 and 8, and has exponential-size. Observe also that $\mathcal{RG}(\mathcal{A})$ is constructed in a forward manner: we start by the initial constrained region (*i.e.* the region of valuation $\mathbf{0}$ with the zero matrix as shrinking constraint), and compute its successors in $\mathcal{RG}(\mathcal{A})$. Then, if Controller has a winning strategy in $\mathcal{RG}(\mathcal{A})$, we construct a winning strategy for $\mathcal{G}_\delta(\mathcal{A})$ by a backward traversal of $\mathcal{RG}(\mathcal{A})$, using Lemmas 7 and 8. Thus, we construct $\mathcal{RG}(\mathcal{A})$ by propagating shrinking constraints forward, but later do a backward traversal in it. The correctness of the construction is stated as follows.

Proposition 9. *Controller has a winning strategy in $\mathcal{RG}(\mathcal{A})$ if, and only if there exists $\delta_0 > 0$ such that Controller wins $\mathcal{G}_\delta(\mathcal{A})$ for all $\delta \in [0, \delta_0]$.*

Note that as we compute a winning strategy for Controller (if any) by Proposition 9, we can also compute a corresponding δ_0. One can show, by a rough estimation, that $1/\delta_0$ is at worst doubly exponential in the size of $\mathcal{A}$.

Let us point out an interesting intermediary result of the proof: given a winning strategy for Perturbator in $\mathcal{RG}(\mathcal{A})$, we show that there is a winning strategy for Perturbator in $\mathcal{G}_\delta(\mathcal{A})$ that keeps the compatible runs close to borders of regions where shrinking constraints are 0.

The upper bound of Theorem 2 is a consequence of the above proposition, since $\mathcal{RG}(\mathcal{A})$ has exponential size and finite reachability games can be solved in time polynomial in the size of the game. The EXPTIME lower bound is obtained by simulating an alternating-time linear-bounded Turing machine. Simulation of the transitions is rather standard in timed-automata literature (though we must be careful here as delays can be perturbed). The difficult point is to simulate conjunctions: this is achieved using the module of Fig. 8. From the initial state, Controller has no choice but to play the first transition when $y = 1$. Perturbator can either anticipate or delay this transition, which will determine which of the dashed or dotted transitions is available next. This way, Perturbator decides by which end the module is exited.

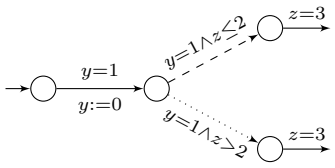


Fig. 8. Conjunction

5 Conclusion

We considered a game-based approach to robust reachability in timed automata. We proved that robust schedulers for reachability objectives can be synthesized, and that the existence of such a scheduler is EXPTIME-complete (hence harder than classical reachability [2]). We are currently working on a zone-based version of the algorithm, and on extending the techniques of this paper to the synthesis of robust controllers in timed games, which will answer an open problem posed in [6] for reachability objectives. Natural further works also include the synthesis of robust schedulers for safety objectives. This seems really challenging, and the abstraction we have built here is not correct in this case (it requires at least a notion of *profitable cycles à la* [7]). Another interesting direction for future work is to assume imprecisions are probabilistic, that is, once Controller has chosen a delay d, the real delay is chosen in a stochastic way in the interval $[d-\delta, d+\delta]$.

References

1. Y. Adbeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *TCS*, 354(2):272–300, 2006.
2. R. Alur and D. L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.
3. E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *SSSC'98*, pp. 469–474. Elsevier, 1998.
4. J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *ACPN'03*, vol. 2098 of *LNCS*, pp. 87–124. Springer, 2004.
5. P. Bouyer, N. Markey, and O. Sankur. Robust reachability in timed automata: A game-based approach. Technical Report LSV-12-07, Lab. Specification & Verification, ENS Cachan, France, May 2012.
6. K. Chatterjee, T. A. Henzinger, and V. S. Prabhu. Timed parity games: Complexity and robustness. *LMCS*, 7(4), 2010.
7. M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin. Robust safety of timed automata. *FMSD*, 33(1-3):45–84, 2008.
8. M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. *FAC*, 17(3):319–341, 2005.
9. R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Applied Maths*, 17(2):416–429, 1969.
10. V. Gupta, T. A. Henzinger, and R. Jagadeesan. Robust timed automata. In *HART'97*, vol. 1201 of *LNCS*, pp. 331–345. Springer, 1997.
11. T. A. Henzinger and J. Sifakis. The embedded systems design challenge. In *FM'06*, vol. 4085 of *LNCS*, pp. 1–15. Springer, 2006.
12. H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, 2011.
13. K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Intl J. STTT*, 1(1-2):134–152, 1997.
14. N. Markey. Robustness in real-time systems. In *SIES'11*, pp. 28–34. IEEE Comp. Soc. Press, 2011.
15. A. Puri. Dynamical properties of timed automata. *DEDS*, 10(1-2):87–113, 2000.
16. J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker. A definition and classification of timing anomalies. In *WCET'06*, 2006.
17. O. Sankur, P. Bouyer, and N. Markey. Shrinking timed automata. In *FSTTCS'11*, vol. 13 of *LIPIcs*, pp. 375–386. LZI, 2011.
18. S. Yovine. Kronos: A verification tool for real-time systems. *Intl J. STTT*, 1(1-2):123–133, 1997.