Fast Algorithms for Handling Diagonal Constraints in Timed Automata

Paul Gastin[1], Sayan Mukherjee[2], and B. Srivathsan[2(✉)]

[1] LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, Cachan, France
paul.gastin@lsv.fr
[2] Chennai Mathematical Institute, Chennai, India
{sayanm,sri}@cmi.ac.in

Abstract. A popular method for solving reachability in timed automata proceeds by enumerating reachable sets of valuations represented as zones. A naïve enumeration of zones does not terminate. Various termination mechanisms have been studied over the years. Coming up with efficient termination mechanisms has been remarkably more challenging when the automaton has diagonal constraints in guards.

In this paper, we propose a new termination mechanism for timed automata with diagonal constraints based on a new simulation relation between zones. Experiments with an implementation of this simulation show significant gains over existing methods.

Keywords: Timed automata · Diagonal constraints · Reachability · Zones · Simulations

1 Introduction

Timed automata have emerged as a popular model for systems with real-time constraints [2]. Timed automata are finite automata extended with real-valued variables called *clocks*. All clocks are assumed to start at 0, and increase at the same rate. Transitions of the automaton can make use of these clocks to disallow behaviours which violate timing constraints. This is achieved by making use of *guards* which are constraints of the form $x \leq 5$, $x - y \geq 3$, $y > 7$, etc. where x, y are clocks. A transition guarded by $x \leq 5$ says that it can be fired only when the value of clock x is ≤ 5. Another important feature is the *reset* of clocks in transitions. Each transition can specify a subset of clocks whose values become 0 once the transition is fired. The combination of guards and resets allows to track timing distance between events. A basic question that forms the core of timed automata technology is *reachability*: given a timed automaton, does there

This work is supported by UMI Relax. The first author is partly supported by ANR project TickTac (ANR-18-CE40-0015) and third author by CEFIPRA project IoTTTA (Indo-French program in ICST-DST/CNRS ref. 2016-01). The second and third authors are partly supported by Infosys Foundation (India) and Tata Consultancy Services - Innovation Labs (Pune, India).

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 41–59, 2019.
https://doi.org/10.1007/978-3-030-25540-4_3

exist an execution from its initial state to a final state. This question is known to be decidable [2]. Various algorithms for this problem have been studied over the years and have been implemented in tools [6,21,26,28,31,32].

Since the clocks are real valued variables, the space of configurations of a timed automaton (consisting of a state and a valuation of the clocks) is infinite and an explicit enumeration is not possible. The earliest solution to reachability was to partition this space into a finite number of *regions* and build a region graph that provides a finite abstraction of the behaviour of the timed automaton [2]. However, this solution was not practical. Subsequent works introduced the use of *zones* [14]. Zones are special sets of clock valuations with efficient data structures and manipulation algorithms [6]. Within zone based algorithms, there is a division: forward analysis versus backward analysis. The current industry strength tool UPPAAL [28] implements a forward analysis approach, as this works better in the presence of other discrete data structures used in UPPAAL models [9]. We focus on this forward analysis approach using zones in this paper.

The forward analysis of a timed automaton essentially enumerates sets of reachable configurations stored as zones. Some extra care needs to be taken for this enumeration to terminate. Traditional development of timed automata made use of *extrapolation* operators over zones to ensure termination. These are functions which map a zone to a bigger zone. Importantly, the range of these functions is finite. The goal was to come up with extrapolation operators which are sound: adding these extra valuations should not lead to new behaviours. This is where the role of *simulations* between configurations was studied and extrapolation operators based on such simulations were devised [14]. A certain extrapolation operation, which is now known as Extra_M [5] was proposed and reachability using Extra_M was implemented in tools [14].

A seminal paper by Bouyer [9] revealed that Extra_M is not correct in the presence of *diagonal constraints* in guards. These are constraints of the form $x - y \triangleleft c$ where $\triangleleft$ is either $<$ or $\leq$, and c is an integer. Moreover, it was proved that no such extrapolation operation would be correct when there are diagonal constraints present. It was shown that for automata without diagonal constraints (henceforth referred to as diagonal-free automata), the extrapolation works. After this result, developments in timed automata reachability focussed on the class of diagonal-free automata [4,5,23,24], and diagonal constraints were mostly sidelined. All these developments have led to quite efficient algorithms for diagonal-free timed automata.

Diagonal constraints are a useful modeling feature and occur naturally in certain problems, especially scheduling [3,17,20,27] and logic-automata translations [16,25], also in [29]. It is however known that they do not add any expressive power: every timed automaton can be converted into a diagonal-free timed automaton [7]. This conversion suffers from an exponential blowup, which was later shown to be unavoidable: diagonal constraints could potentially give exponentially more succinct models [10]. Therefore, a good forward analysis algorithm that works directly on a timed automaton with diagonal constraints would be handy. This is the subject of this paper.

Related Work. The first attempt at such an algorithm was to split the (extrapolated) zones with respect to the diagonal constraints present in the automaton [6]. This gave a correct procedure, but since zones are split, an enumeration starts from each small zone leading to an exponential blow-up in the number of visited zones. A second attempt was to do a more refined conversion into a diagonal free automaton by detecting "relevant" diagonals [13,30] in an iterative manner. In order to do this, special data structures storing sets of sets of diagonal constraints were utilized. In [18] we extended the works [5] and [23] on diagonal-free automata to the case of diagonal constraints. All the approaches suffer from either a space or time bottleneck and are incomparable to the efficiency and scalability of tools for diagonal-free automata.

Our Contributions. The goal of this paper is to come up with fast algorithms for handling diagonal constraints. Since the extrapolation based approach is a dead end, we work with simulation between zones directly, as in [23] and [18]. We propose a new simulation relation between zones that is correct in the presence of diagonal constraints (Sect. 3). We give an algorithm to test this simulation between zones (Sect. 4). We have incorporated this simulation test in (an older version of) the tool TChecker [21] checking reachability for timed automata, and compared our results with the state-of-the-art tool UPPAAL. Experiments show an encouraging gain, both in the number of zones enumerated and in the time taken by the algorithm, sometimes upto four orders of magnitude (Sect. 6). The main advantage of our approach is that it does not split zones, and furthermore it leverages the optimizations studied for diagonal-free automata.

From a technical point of view, our presentation does not make use of regions and instead works with valuations, zones and simulation relations. We think that this presentation provides a clearer perspective - as a justification of this claim, we extend our simulation to timed automata with general updates of the form $x := c$ and $x := y + d$ in transitions (where x, y are clocks and c, d are constants) in a rather natural manner (Sect. 5). In general, reachability for timed automata with updates is undecidable [12]. Some decidable cases have been proposed for which the algorithms are based on regions. For decidable subclasses containing diagonal constraints, no zone based approach has been studied. Our proposed method includes these classes, and also benefits from zones and standard optimizations studied for diagonal-free automata.

Missing proofs can be found in the full version of this paper [19].

2 Preliminaries

Let $\mathbb{N}$ be the set of natural numbers, $\mathbb{R}_{\geq 0}$ the set of non-negative reals and $\mathbb{Z}$ the set of integers. Let X be a finite set of variables ranging over $\mathbb{R}_{\geq 0}$, called *clocks*. Let $\Phi(X)$ denote the set of constraints φ formed using the following grammar: $\varphi := x \lhd c \mid c \lhd x \mid x - y \lhd d \mid \varphi \wedge \varphi$, where $x, y \in X$, $c \in \mathbb{N}$, $d \in \mathbb{Z}$ and $\lhd \in \{<, \leq\}$. Constraints of the form $x \lhd c$ and $c \lhd x$ are called *non-diagonal constraints* and those of the form $x - y \lhd c$ are called *diagonal constraints*. We have adopted a convention that in non-diagonal constraints $x \lhd c$ and $c \lhd x$, the

constant c is restricted to $\mathbb{N}$. A *clock valuation* v is a function which maps every clock $x \in X$ to a real number $v(x) \in \mathbb{R}_{\geq 0}$. A valuation is said to satisfy a guard g, written as $v \models g$ if replacing every x in g with $v(x)$ makes the constraint g true. For $\delta \in \mathbb{R}_{\geq 0}$ we write $v + \delta$ for the valuation which maps every x to $v(x) + \delta$. Given a subset of clocks $R \subseteq X$, we write $[R]v$ for the valuation which maps each $x \in R$ to 0 and each $x \notin R$ to $v(x)$.

A *timed automaton* $\mathcal{A}$ is a tuple (Q, X, q_0, T, F) where Q is a finite set of states, X is a finite set of clocks, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of accepting states and $T \in Q \times \Phi(X) \times 2^X \times Q$ is a set of transitions. Each transition $t \in T$ is of the form (q, g, R, q') where q and q' are respectively the source and target states, g is a constraint called the *guard*, and R is a set of clocks which are *reset* in t. We call a timed automaton *diagonal-free* if guards in transitions do not use diagonal constraints.

A *configuration* of $\mathcal{A}$ is a pair (q, v) where $q \in Q$ and v is a valuation. The semantics of a timed automaton is given by a transition system $\mathcal{S}_\mathcal{A}$ whose states are the configurations of $\mathcal{A}$. Transitions in $\mathcal{S}_\mathcal{A}$ are of two kinds: *delay* transitions are given by $(q, v) \xrightarrow{\delta} (q, v + \delta)$ for all $\delta \geq 0$, and *action* transitions are given by $(q, v) \xrightarrow{t} (q', v')$ for each $t := (q, g, R, q')$, if $v \models g$ and $v' = [R]v$. We write $\xrightarrow{\delta,t}$ for a sequence of delay δ followed by action t. A run of $\mathcal{A}$ is an alternating sequence of delay-action transitions starting from the initial state q_0 and the initial valuation $\mathbf{0}$ which maps every clock to 0: $(q_0, \mathbf{0}) \xrightarrow{\delta_0,t_0} (q_1, v_1) \xrightarrow{\delta_1,t_1} \cdots (q_n, v_n)$. A run of the above form is said to be accepting if the last state $q_n \in F$. The *reachability problem* for timed automata is the following: given an automaton $\mathcal{A}$, decide if there exists an accepting run. This problem is known to be PSPACE-complete [2]. Since the semantics $\mathcal{S}_\mathcal{A}$ is infinite, solutions to the reachability problem work with a finite abstraction of $\mathcal{S}_\mathcal{A}$ that is sound and complete. Before we explain one of the popular solutions to reachability, we state a result which allows to convert every timed automaton into a diagonal-free timed automaton.

Theorem 1. *[7] For every timed automaton $\mathcal{A}$, there exists a diagonal-free timed automaton $\mathcal{A}_{df}$ s.t. there is a bijection between runs of $\mathcal{A}$ and $\mathcal{A}_{df}$. The number of states in $\mathcal{A}_{df}$ is $2^d \cdot n$ where d is the number of diagonal constraints and n is the number of states of $\mathcal{A}$.*

The above theorem allows to solve the reachability of a timed automaton $\mathcal{A}$ by first converting it into the diagonal free automaton $\mathcal{A}_{df}$ and then checking reachability on $\mathcal{A}_{df}$. However, this conversion comes with a systematic exponential blowup (in terms of the number of diagonal constraints present in $\mathcal{A}$). It was shown in [10] that such a blowup is unavoidable in general. We will now recall the general algorithm for analyzing timed automata, and then move into specific details which depend on whether the automaton has diagonal constraints or not.

Zones and Simulations. Fix a timed automaton $\mathcal{A}$ with clock set X for the rest of the discussion in this section. As the space of valuations of $\mathcal{A}$ is infinite, algorithms work with sets of valuations called *zones*. A zone is set of clock valuations given by a conjunction of constraints of the form $x - y \lhd c$, $x \lhd c$ and

$c \triangleleft x$ where $c \in \mathbb{Z}$ and $\triangleleft \in \{<, \leq\}$, for example the solutions of $x - y < 5 \wedge y \leq 10$ is a zone. The transition relation over configurations (q, v) is extended to (q, Z) where Z is a zone. We define the following operations on zones given a guard g and a set of clocks R: time elapse $\overrightarrow{Z} = \{v + \delta \mid v \in Z, \delta \geq 0\}$; guard intersection $Z \wedge g := \{v \mid v \in Z \text{ and } v \models g\}$ and reset $[R]Z := \{[R]v \mid v \in Z\}$. It can be shown that all these operations result in zones. Zones can be efficiently represented and manipulated using Difference Bound Matrices (DBMs) [15].

The *zone graph* $ZG(\mathcal{A})$ of timed automaton $\mathcal{A}$ is a transition system whose nodes are of the form (q, Z) where q is a state of $\mathcal{A}$ and Z is a zone. For each transition $t := (q, g, R, q')$ of $\mathcal{A}$, and each zone (q, Z) there is a transition $(q, Z) \Rightarrow^t (q', Z')$ where $Z' = \overrightarrow{[R](Z \wedge g)}$. The initial node is (q_0, Z_0) where q_0 is the initial state of $\mathcal{A}$ and $Z_0 = \{\mathbf{0} + \delta \mid \delta \geq 0\}$ is the zone obtained by elapsing an arbitrary delay from the initial valuation. A path in the zone graph is a sequence $(q_0, Z_0) \Rightarrow^{t_0} (q_1, Z_1) \Rightarrow^{t_1} \cdots \Rightarrow^{t_{n-1}} (q_n, Z_n)$ starting from the initial node. The path is said to be accepting if q_n is an accepting state. The zone graph is known to be sound and complete for reachability.

Theorem 2. *[14] $\mathcal{A}$ has an accepting run iff $ZG(\mathcal{A})$ has an accepting path.*

This does not yet give an algorithm as the zone graph $ZG(\mathcal{A})$ is still not finite. Moreover, there are examples of automata for which the reachable part of $ZG(\mathcal{A})$ is also infinite: starting from the initial node, applying the successor computation leads to infinitely many zones. Two different approaches have been studied to get finiteness, both of them based on the usage of *simulation relations*.

A (time-abstract) simulation relation ($\preccurlyeq$) between configurations of $\mathcal{A}$ is a reflexive and transitive relation such that $(q, v) \preccurlyeq (q', v')$ implies $q = q'$ and (1) for every $\delta \geq 0$, there exists $\delta' \geq 0$ such that $(q, v + \delta) \preccurlyeq (q, v' + \delta')$ and (2) for every transition t of $\mathcal{A}$, if $(q, v) \xrightarrow{t} (q_1, v_1)$ then $(q, v') \xrightarrow{t} (q_1, v_1')$ such that $(q_1, v_1) \preccurlyeq (q_1, v_1')$.

We say $v \preccurlyeq v'$, read as v is simulated by v' if $(q, v) \preccurlyeq (q, v')$ for all states q. The simulation relation can be extended to zones: $Z \preccurlyeq Z'$ if for every $v \in Z$ there exists $v' \in Z'$ such that $v \preccurlyeq v'$. We write $\downarrow Z$ for $\{v \mid \exists v' \in Z \text{ s.t. } v \preccurlyeq v'\}$. The simulation relation $\preccurlyeq$ is said to be finite if the function mapping zones Z to the down sets $\downarrow Z$ has finite range. We now recall a specific simulation relation $\preccurlyeq_{LU}$ [5,23]. Current algorithms and tools for diagonal-free automata are based on this simulation. The conditions required for $v \preccurlyeq_{LU} v'$ ensure that when all lower bound constraints $c \triangleleft x$ satisfy $c \leq L(x)$ and all upper bound constraints $x \triangleleft c$ satisfy $c \leq U(x)$, whenever v satisfies a constraint, v' will also satisfy it.

Definition 1 (LU-bounds and the relation $\preccurlyeq_{LU}$ [5,23]). *An LU-bounds function is a pair of functions $L : X \mapsto \mathbb{N} \cup \{-\infty\}$ and $U : X \mapsto \mathbb{N} \cup \{-\infty\}$ that map each clock to either a non-negative constant or $-\infty$. Given an LU-bounds function, we define $v \preccurlyeq_{LU} v'$ for valuations v, v' if for every clock $x \in X$:*

$$v'(x) < v(x) \text{ implies } L(x) < v'(x) \quad \text{and} \quad v(x) < v'(x) \text{ implies } U(x) < v(x).$$

Reachability in Diagonal-Free Timed Automata. A natural method to get finiteness of the zone graph is to prune the zone graph computation through simulations $Z \preccurlyeq Z'$: do not explore a node (q, Z) if there is an already visited node (q, Z') such that $Z \preccurlyeq Z'$. Since these simulation tests need to be done often during the zone graph computation, an efficient algorithm for performing this test is crucial. Note that $Z \preccurlyeq Z'$ iff $Z \subseteq \downarrow Z'$. However, it is known that the set $\downarrow Z'$ is not necessarily a zone (this was proved for $\downarrow_{LU} Z'$ in [5]), and hence no simple zone inclusions are applicable. The first algorithms for timed automata followed a different approach, which we call the *extrapolation* approach. In this approach, whenever a new zone Z is discovered by the algorithm, a new zone $\mathsf{Extra}(Z)(\supseteq Z)$ gets computed and stored in the place of Z.

Reachability Algorithm Using Zone Extrapolation. The input to the algorithm is a timed automaton $\mathcal{A}$. The algorithm maintains two lists, Passed and Waiting. Initially, the node $(q_0, \mathsf{Extra}(Z_0))$ is added to the Waiting list (recall that (q_0, Z_0) is the initial node of the zone graph $ZG(\mathcal{A})$). Wlog. we assume that q_0 is not accepting. The algorithm repeatedly performs the following steps:

Step 1. If Waiting is empty, then return "$\mathcal{A}$ has no accepting run"; else pick (and remove) a node (q, Z) from Waiting. Add (q, Z) to Passed.

Step 2. For each transition $t := (q, g, R, q_1)$, compute the successor $(q, Z) \Rightarrow^t (q_1, Z_1)$: if $Z_1 \neq \emptyset$ perform the following operations - if q_1 is accepting, return "$\mathcal{A}$ has an accepting run"; else compute $\hat{Z}_1 := \mathsf{Extra}(Z_1)$ and check if there exists a node (q_1, Z_1') in Passed or Waiting such that $\hat{Z}_1 \subseteq Z_1'$: if yes, ignore the node $(q_1, \hat{Z}_1)$, otherwise add $(q_1, \hat{Z}_1)$ to Waiting.

Several extrapolation operators (Extra_M, Extra_{LU}, Extra^+_{LU}) were introduced in [5]. The function Extra^+_{LU} has nice properties - (1) $\mathsf{Extra}^+_{LU}(Z) \subseteq \downarrow_{LU} Z$ and (2) $\mathsf{Extra}^+_{LU}(Z)$ is a zone for all Z. These properties give an algorithm that performs only efficient zone operations: successor computations and zone inclusions.

Reachability Algorithm Using Simulations. The initial node (q_0, Z_0) is added to the Waiting list. Wlog. we assume that q_0 is not accepting. The algorithm repeatedly performs the following steps:

Step 1. If Waiting is empty, then return "$\mathcal{A}$ has no accepting run"; else pick (and remove) a node (q, Z) from Waiting. Add (q, Z) to Passed.

Step 2. For each transition $t := (q, g, R, q_1)$, compute the successor $(q, Z) \Rightarrow^t (q_1, Z_1)$: if $Z_1 \neq \emptyset$ perform the following operations - if q_1 is accepting, return "$\mathcal{A}$ has an accepting run"; else check if there exists a node (q_1, Z_1') in Passed or Waiting such that $Z_1 \preccurlyeq Z_1'$: if yes, ignore the node (q_1, Z_1), otherwise add (q_1, Z_1) to Waiting.

An $\mathcal{O}(|X|^2)$ algorithm for $Z \preccurlyeq_{LU} Z'$ was proposed in [23]. The efficiency of this simulation check makes it well suited for use in practice. Moreover, as $\mathsf{Extra}^+_{LU}(Z) \subseteq \downarrow_{LU} Z$, we expect to get more simulations (and hence quicker termination) through $\preccurlyeq_{LU}$.

Reachability in the Presence of Diagonal Constraints. The $\preccurlyeq_{LU}$ relation is no longer a simulation when diagonal constraints are present. Moreover, it was shown in [9] that no extrapolation operator (along the lines of Extra^{+}_{LU}) can work in the presence of diagonal constraints. The first option to deal with diagonals is to use Theorem 1 to get a diagonal free automaton and then apply the methods discussed previously. One problem with this is the systematic exponential blowup introduced in the number of states of the resulting automaton. Another problem is to get diagnostic information: counterexamples need to be translated back to the original automaton [6]. Various methods have been studied to circumvent the diagonal free conversion and instead work on the automaton with diagonal constraints directly. We recall the approach used in the state-of-the-art tool UPPAAL below.

Zone Splitting [6]. The paper introducing timed automata gave a notion of equivalence between valuations $v \simeq_M v'$ parameterized by a function M mapping each clock x to the maximum constant M among the guards of the automaton that involve x. This equivalence is a finite simulation for diagonal-free automata. Equivalence classes of $\simeq_M$ are called regions. This was extended to the diagonal case by [6] as: $v \simeq^d_M v'$ if $v \simeq_M v'$ and for all diagonal constraints g present in the automaton, if $v \models g$ then $v' \models g$. The $\simeq^d_M$ relation splits the regions further, such that each region is either entirely included inside g, or entirely outside g for each g. The next step is to use this notion of equivalence in zones. The paper [6] follows the extrapolation approach: to each zone Z, an extrapolation operation $\mathsf{Extra}_M(Z)$ is applied; this adds some valuations which are $\simeq_M$ equivalent to valuations in Z; then it is further split into multiple zones, so that each small zone is either inside g or outside g for each diagonal constraint g. If d is the number of diagonal constraints present in the automaton, this splitting process can give rise to 2^d zones for each zone Z. From each small zone, the zone graph computation is started. Essentially, the exponential blow-up at the state level which appeared in the diagonal-free conversion now appears in the zone level.

In this paper, we propose a new simulation to handle diagonal constraints. This has two advantages - using this avoids the blow-up in the number of nodes arising due to zone splitting, and the simulation test between zones has an efficient implementation and is significantly quicker than the simulation of [18].

3 A New Simulation Relation

We start with a definition of a relation between timed automata configurations, which in some sense "declares" upfront what we need out of a simulation relation that can be used in a reachability algorithm. As we proceed, we will make its description more concrete and give an effective simulation algorithm between zones, that can be implemented. Fix a clock set X. This generates constraints $\Phi(X)$.

Definition 2 (the relation $\sqsubseteq_{\mathcal{G}}$). *Let $\mathcal{G}$ be a (finite or infinite) set of constraints. We say $v \sqsubseteq_{\mathcal{G}} v'$ if for all $\varphi \in \mathcal{G}$ and all $\delta \geq 0$, $v + \delta \models \varphi$ implies $v' + \delta \models \varphi$.*

Our goal is to utilize the above relation in a simulation (as defined in p. xx) for a timed automaton. Directly from the definition, we get the following lemma which shows that the $\sqsubseteq_{\mathcal{G}}$ relation is preserved under time elapse.

Lemma 1. *If $v \sqsubseteq_{\mathcal{G}} v'$, then $v + \delta \sqsubseteq_{\mathcal{G}} v' + \delta$ for all $\delta \geq 0$.*

The other kind of transformation over valuations is resets. Given sets of guards $\mathcal{G}_1$, $\mathcal{G}$ and a set of clocks R, we want to find conditions on $\mathcal{G}_1$ and $\mathcal{G}$ so that if $v \sqsubseteq_{\mathcal{G}_1} v'$ then $[R]v \sqsubseteq_{\mathcal{G}} [R]v'$. To do this, we need to answer this question: what guarantees should we ensure for v, v' (via $\mathcal{G}_1$) so that $[R]v \sqsubseteq_{\mathcal{G}} [R]v'$. This motivates the next definition.

Definition 3 (weakest pre-condition of $\sqsubseteq_{\mathcal{G}}$ over resets). *For a constraint φ and a set of clocks R, we define a set of constraints $\mathrm{wp}(\sqsubseteq_{\varphi}, R)$ as follows: when φ is of the form $x \lhd c$ or $c \lhd x$, then $\mathrm{wp}(\sqsubseteq_{\varphi}, R)$ is empty if $x \in R$ and is $\{\varphi\}$ otherwise; when φ is a diagonal constraint $x - y \lhd c$, then $\mathrm{wp}(\sqsubseteq_{\varphi}, R)$ is:*

- $\{x - y \lhd c\}$ *if* $\{x, y\} \cap R = \emptyset$
- $\{x \lhd c\}$ *if* $y \in R$, $x \notin R$ *and* $c \geq 0$
- $\{-c \lhd y\}$ *if* $x \in R$, $y \notin R$ *and* $-c \geq 0$
- *empty, otherwise.*

For a set of guards $\mathcal{G}$, we define $\mathrm{wp}(\sqsubseteq_{\mathcal{G}}, R) := \bigcup_{\varphi \in \mathcal{G}} \mathrm{wp}(\sqsubseteq_{\varphi}, R)$.

Note that the relation $\sqsubseteq_{\mathcal{G}}$ is parameterized by a set of constraints. Additionally, we desire this set to be finite, so that the relation can be used in an algorithm. We need to first link an automaton $\mathcal{A}$ with such a set of constraints. One way to do it is to take the set of all guards present in the automaton and to close it under weakest pre-conditions with respect to all possible subsets of clocks. A better approach is to consider a set of constraints for each state, as in [4] where the parameters for extrapolation (the maximum constants appearing in guards) are calculated at each state.

Definition 4 (State based guards). *Let $\mathcal{A} = (Q, X, q_0, T, F)$ be a timed automaton. We associate a set of guards $\mathcal{G}(q)$ for each state $q \in Q$, which is the least set of guards (for the coordinate-wise subset inclusion order) such that for every transition (q, g, R, q_1): the guard g and the set $\mathrm{wp}(\sqsubseteq_{\mathcal{G}(q_1)}, R)$ are present in $\mathcal{G}(q)$. More precisely, $\{\mathcal{G}(q)\}_{q \in Q}$ is the least solution to the following set of equations written for each $q \in Q$:*

$$\mathcal{G}(q) = \bigcup_{(q,g,R,q_1) \in T} \{g\} \cup \mathrm{wp}(\sqsubseteq_{\mathcal{G}(q_1)}, R)$$

All constraints present in the set $\mathrm{wp}(\sqsubseteq_{\mathcal{G}(q_1)}, R)$ contain constants which are already present in $\sqsubseteq_{\mathcal{G}(q_1)}$. The least solution to the above set of equations can therefore be obtained by a fixed point computation which starts with $\mathcal{G}(q)$ set to $\bigcup_{(q,g,R,q_1) \in T} \{g\}$ and then repeatedly updates the weakest-preconditions. Since no new constants are generated in this process, the fixed point computation terminates. We now have the ingredients to define a simulation relation over configurations of a timed automaton with diagonal constraints.

Definition 5 ($\mathcal{A}$-simulation). *Let $\mathcal{A} = (Q, X, q_0, T, F)$ be a timed automaton and let the set of guards $\mathcal{G}(q)$ of Definition 4 be associated to every state $q \in Q$. We define a relation $\preccurlyeq_{\mathcal{A}}$ between configurations of $\mathcal{A}$ as $(q, v) \preccurlyeq_{\mathcal{A}} (q, v')$ if $v \sqsubseteq_{\mathcal{G}(q)} v'$.*

Lemma 2. *The relation $\preccurlyeq_{\mathcal{A}}$ is a simulation on the configurations of timed automaton $\mathcal{A}$.*

As pointed before, Definition 2 gives a declarative description of the simulation and it is unclear how to work with it algorithmically, even when the set of constraints $\mathcal{G}$ is finite. The main issue is with the $\forall\delta$ quantification, which is not finite. We will first provide a characterization that brings out the fact that this $\forall\delta$ quantification is irrelevant for diagonal constraints (essentially because value of $v(x) - v(y)$ does not change with time elapse). Given a set of constraints $\mathcal{G}$, let $\mathcal{G}^- \subseteq \mathcal{G}$ be the set of non-diagonal constraints in $\mathcal{G}$.

Proposition 1. *$v \sqsubseteq_{\mathcal{G}} v'$ iff $v \sqsubseteq_{\mathcal{G}^-} v'$ and for all diagonal constraints $\varphi \in \mathcal{G}$, if $v \models \varphi$ then $v' \models \varphi$.*

It now amounts to solving the $\forall\delta$ problem for non-diagonals. It turns out that the $\preccurlyeq_{LU}$ simulation achieves this, almost. We will see this in more detail in the next section.

4 Algorithm for $Z \sqsubseteq_{\mathcal{G}} Z'$

Fix a finite set of guards $\mathcal{G}$. Restating the definition of $\sqsubseteq_{\mathcal{G}}$ extended to zones: $Z \sqsubseteq_{\mathcal{G}} Z'$ if for all $v \in Z$ there exists a $v' \in Z'$ such that $v \sqsubseteq_{\mathcal{G}} v'$. In this section, we will view the characterization of $\sqsubseteq_{\mathcal{G}}$ as in Proposition 1 and give an algorithm to check $Z \sqsubseteq_{\mathcal{G}} Z'$ that uses as an oracle a test $Z \sqsubseteq_{\mathcal{G}^-} Z'$. We discuss the computation of $Z \sqsubseteq_{\mathcal{G}^-} Z'$ later in this section. We start with an observation following from Proposition 1.

Lemma 3. *Let $\varphi := x - y \lhd c$ be a diagonal constraint in $\mathcal{G}$. Then $Z \sqsubseteq_{\mathcal{G}} Z'$ if and only if $Z \cap \varphi \sqsubseteq_{\mathcal{G}'} Z' \cap \varphi$ and $Z \cap \neg\varphi \sqsubseteq_{\mathcal{G}'} Z'$ where $\mathcal{G}' = \mathcal{G} \setminus \{\varphi\}$.*

If $\mathcal{G}$ has no diagonal constraints, $Z \sqsubseteq_{\mathcal{G}} Z'$ if and only if $Z \sqsubseteq_{\mathcal{G}^-} Z'$.

This leads to the following algorithm consisting of two mutually recursive procedures. This algorithm is essentially an implementation of the above lemma, with two optimizations:

- we start with the non-diagonal check in Line 6 of Algorithm 1 - if this is already violated, then the algorithm returns false;
- suppose $Z \sqsubseteq_{\mathcal{G}^-} Z'$, the next task is to perform the checks in the first statement of Lemma 3 - this is done by Algorithm 2; note however that when Algorithm 2 is called, we already have $Z \sqsubseteq_{\mathcal{G}^-} Z'$, hence $Z \cap \neg\varphi \sqsubseteq_{\mathcal{G}^-} Z'$. Therefore we use an optimization in Line 7 by calling Algorithm 2 directly (as the check in Line 6 of Algorithm 1 will be redundant).

```
1 check Z ⊑_G Z':
2     if Z = ∅ :
3         return true
4     if Z' = ∅ :
5         return false
6     if Z ⋢_{G⁻} Z' :
7         return false
8     return Z ⊑*_G Z'
```

Algorithm 1

```
1 check Z ⊑*_G Z':
2     if G does not contain any
      diagonal constraints :
3         return true
4     pick a diagonal constraint
      φ = x − y ◁ c from G
5     G' ⟵ G \ {φ}
6     if Z ∩ ¬φ ≠ ∅ :
7         if Z ∩ ¬φ ⋢*_{G'} Z' :
8             return false
9     return Z ∩ φ ⊑_{G'} Z' ∩ φ
```

Algorithm 2

Computing $Z \sqsubseteq_{\mathcal{G}^-} Z'$. We will use $\preccurlyeq_{LU}$ to approximate $\sqsubseteq_{\mathcal{G}^-}$: in our implementation of the above algorithms, we replace $Z \sqsubseteq_{\mathcal{G}^-} Z'$ with $Z \preccurlyeq_{LU} Z'$. This works because for an appropriate choice of LU (explained below), we have $Z \preccurlyeq_{LU(\mathcal{G})} Z' \Rightarrow Z \sqsubseteq_{\mathcal{G}^-} Z'$. The converse is not true as the LU bounds functions cannot distinguish between guards with $<$ and $\leq$ comparisons. Therefore, the $\preccurlyeq_{LU}$ simulation does not characterize $v \sqsubseteq_{\mathcal{G}^-} v'$ completely. Although we are aware of the (rather technical) modifications to $\preccurlyeq_{LU}$ simulation that are needed for this characterization, we choose to use the existing $\preccurlyeq_{LU}$ directly as it is safe to do so and it has already been implemented in tools. This gives us a finer simulation than $v \sqsubseteq_{\mathcal{G}^-} v'$.

Definition 6 (LU-bounds from $\mathcal{G}$). *Let $\mathcal{G}$ be a finite set of constraints. We define $LU(\mathcal{G})$ to denote the pair of functions $L_{\mathcal{G}}$ and $U_{\mathcal{G}}$ defined as follows:*

$$L_{\mathcal{G}}(x) = \begin{cases} -\infty & \text{if there is no guard of the form } c \lhd x \text{ in } \mathcal{G} \\ \max\{c \mid c \lhd x \in \mathcal{G}\} & \text{otherwise} \end{cases}$$

$$U_{\mathcal{G}}(x) = \begin{cases} -\infty & \text{if there is no guard of the form } x \lhd c \text{ in } \mathcal{G} \\ \max\{c \mid x \lhd c \in \mathcal{G}\} & \text{otherwise} \end{cases}$$

Lemma 4. *For every set of constraints $\mathcal{G}$, $v \preccurlyeq_{LU(\mathcal{G})} v'$ implies $v \sqsubseteq_{\mathcal{G}^-} v'$.*

The above observations call for the next definition and subsequent lemmas.

Definition 7 (approximating $\sqsubseteq_{\mathcal{G}}$). *Let $\mathcal{G}$ be a finite set of constraints. We define a relation $\sqsubseteq^{LU}_{\mathcal{G}}$ as follows: $v \sqsubseteq^{LU}_{\mathcal{G}} v'$ if $v \preccurlyeq_{LU(\mathcal{G})} v'$ and for all diagonal constraints $\varphi \in \mathcal{G}$, if $v \models \varphi$ then $v' \models \varphi$. Similarly, define $\preccurlyeq^{LU}_{\mathcal{A}}$ as $(q, v) \preccurlyeq^{LU}_{\mathcal{A}} (q, v')$ if $v \sqsubseteq^{LU}_{\mathcal{G}(q)} v'$.*

Lemma 5. *The relation $\preccurlyeq^{LU}_{\mathcal{A}}$ is a finite simulation on the configurations of $\mathcal{A}$.*

The above lemma and the fact that $Z \preccurlyeq_{LU(\mathcal{G})} Z'$ can be checked in $\mathcal{O}(|X|^2)$ [23,33], imply the following theorem.

Theorem 3. *When using $Z \preccurlyeq_{LU(\mathcal{G})} Z'$ in the place of $Z \sqsubseteq_{\mathcal{G}^-} Z'$, the algorithm is correct and it terminates in $\mathcal{O}(2^d \cdot |X|^2)$ where d is the number of diagonal guards in $\mathcal{G}$.*

From a complexity viewpoint, this algorithm is not efficient since it makes an exponential number of calls in the number of diagonal constraints (in fact this may not be avoidable due to Lemma 6, which follows from the NP-hardness result in [18]). Although the above algorithm does involve many calls, the internal operations involved in each call are simple zone manipulations. Moreover, the preliminary checks (for instance line 6 of Algorithm 1) cut short the number of calls. This is visible in our experiments which are very good, especially with respect to running time, as compared to other methods. A similar hardness was shown for a different simulation in [18], but the implementation there indeed witnessed the hardness, as the time taken by that algorithm was unsatisfactory.

Lemma 6. *Deciding $Z \not\sqsubseteq^{LU}_{\mathcal{G}} Z'$ is NP-complete.*

5 Simulations for Updatable Timed Automata

In the timed automata considered so far, clocks are allowed to be reset to 0 along transitions. We consider in this section more sophisticated transformations to clocks in transitions. These are called *updates*. An update $up : \mathbb{R}^{|X|}_{\geq 0} \mapsto \mathbb{R}^{|X|}$ is a function mapping non-negative $|X|$-dimensional reals (valuations) v to general $|X|$-dimensional reals (which may apriori not be valuations as the coordinates may be negative). The syntax of the update function up is given by a set of atomic updates up_x to each $x \in X$, which are of the form $x := c$ or $x := y + d$ where $c \in \mathbb{N}$, $d \in \mathbb{Z}$ and $y \in X$ (possibly equal to x). Note that we want d to be an integer, since we allow for decrementing clocks, and on the other hand $c \in \mathbb{N}$ since we have non-negative clocks. Given a valuation v and an update up, the valuation $up(v)$ is:

$$up(v)(x) := \begin{cases} c & \text{if } up_x \text{ is } x := c \\ v(y) + d & \text{if } up_x \text{ is } x := y + d \end{cases}$$

Note that in general, due to the presence of updates $x := y+d$, the update $up(v)$ may not yield a clock valuation. However, when it does give a valuation, it can be used as a transformation in timed automata transitions. We say $up(v) \geq 0$ if $up(v)(x) \geq 0$ for all clocks $x \in X$.

An *updateable timed automaton (UTA)* $\mathcal{A} = (Q, X, q_0, T, F)$ is an extension of a classic timed automaton with transitions of the form (q, g, up, q') where up is an update. Semantics extend in the natural way: delay transitions remain the same, and for action transitions $t := (q, g, up, q')$ we have $(q, v) \xrightarrow{t} (q', v')$ if $v \models g$, $up(v) \geq 0$, and $v' = up(v)$. We allow the transition only if the update results

in a valuation. The reachability problem for these automata is known to be undecidable in general [12]. Various subclasses with decidable reachability have been discussed in the same paper. Decidability proofs in [12] take the following flavour, for a given automaton $\mathcal{A}$: (1) divide the space of all valuations into a finite number of equivalence classes called *regions* (2) to build the parameters for the equivalence, derive a set of diophantine equations from the guards of $\mathcal{A}$; if they have a solution then construct the quotient graph of the equivalence (called region graph) parameterized by the obtained solution and check reachability on it; if the equations have no solution, output that reachability for $\mathcal{A}$ cannot be answered. Sufficient conditions on the nature of the updates that give a solution to the diophantine equations have been tabulated in [12]. When the automaton is diagonal-free, the "region-equivalence" can be used to build an extrapolation operation which in turn can be used in a reachability algorithm with zones. When the automaton contains diagonals, the region-equivalence is used to only build a region graph - no effective zone based approach has been studied.

We use a similar idea, but we have two fundamental differences: (1) we want to obtain reachability through the use of simulations on zones, and (2) we build equations over sets of guards as in Definition 4. The advantage of this approach is that this allows the use of coarser simulations over zones. Even for automata with diagonal constraints and updates, we get a zone based algorithm, instead of resorting to regions which are not efficient in practice.

The notion of simulations as in p. xx remains the same, now using the semantics of transitions with updates. We will re-use the simulation relation $\sqsubseteq_{\mathcal{G}}$. We need to extend Definition 3 to incorporate updates. We do this below. Here is a notation: for an update function up, we write $up(x)$ to be c if up_x is $x := c$, and $up(x)$ to be $y + c$ if up_x is $x := y + c$.

Definition 8 (weakest pre-condition of $\sqsubseteq_{\mathcal{G}}$ over updates).
Let up be an update.

For a constraint φ of the form $x \lhd c$ or $c \lhd x$, we define $\mathrm{wp}(\sqsubseteq_\varphi, up)$ *to be respectively* $\{up(x) \lhd c\}$ *or* $\{c \lhd up(x)\}$ *if these resulting constraints are of the form $z \lhd d$ or $d \lhd z$ with $z \in X$ and $d \geq 0$, otherwise* $\mathrm{wp}(\sqsubseteq_\varphi, up)$ *is empty.*

For a constraint $\varphi : x - y \lhd c$, we define $\mathrm{wp}(\sqsubseteq_\varphi, up)$ *to be* $\{up(x) - up(y) \lhd c\}$ *if this constraint is either a diagonal using different clocks, or it is of the form $z \lhd d$ or $d \lhd z$ with $d \geq 0$, otherwise* $\mathrm{wp}(\sqsubseteq_\varphi, up)$ *is empty.*

For a set of guards $\mathcal{G}$, we define $\mathrm{wp}(\sqsubseteq_{\mathcal{G}}, up) := \bigcup_{\varphi \in \mathcal{G}} \mathrm{wp}(\sqsubseteq_\varphi, up)$.

Some examples: $\mathrm{wp}(x \leq 5, x := x + 10)$ is empty, since $up(x)$ is $x + 10$, and the guard $x + 10 \leq 5$ is not satisfiable; $\mathrm{wp}(x \leq 5, x := x - 10)$ is $x \leq 15$, $\mathrm{wp}(x \leq 5, x := c)$ is empty, $\mathrm{wp}(x - y \leq 5, \langle x := z_1, y := z_2 + 10\rangle)$ will be $z_1 - (z_2 + 10) \leq 5$, giving the constraint $z_1 - z_2 \leq 15$, $\mathrm{wp}(x - y \leq 5, \langle x := z + c_1, y := z + c_2\rangle)$ is empty, $\mathrm{wp}(x - y \leq 5, \langle x := c_1, y := z + c_2\rangle)$ is $c = c_1 - 5 - c_2 \leq z$ if $c \geq 0$ and is empty otherwise.

Definition 9 (State based guards). *Let $\mathcal{A} = (Q, X, q_0, T, F)$ be a UTA. We associate a set of constraints $\mathcal{G}(q)$ for each state $q \in Q$, which is the least set of constraints (for the coordinate-wise subset inclusion order) such that for*

every transition (q, g, up, q_1)*: the guard* g *and the set* $\mathrm{wp}(\sqsubseteq_{\mathcal{G}(q_1)}, up)$ *are present in* $\mathcal{G}(q)$*, and in addition constraints that allow the update to happen are also present in* $\mathcal{G}$*. The last condition is given by the weakest precondition of the set of constraints* $\{x \geq 0 \mid x \in X\}$*. Overall,* $\{\mathcal{G}(q)\}_{q\in Q}$ *is the least solution to the following set of equations, for each* $q \in Q$*:*

$$\mathcal{G}(q) = \bigcup_{(q,g,up,q_1)\in T} \left(\{g\} \cup \mathrm{wp}(\sqsubseteq_{\{x\geq 0 | x\in X\}}, up) \cup \mathrm{wp}(\sqsubseteq_{\mathcal{G}(q_1)}, up) \right)$$

The least solution $\{\mathcal{G}(q)\}_{q\in Q}$ *is said to be finite if each* $\mathcal{G}(q)$ *is a finite set of constraints.*

In contrast to the simple reset case, the above set of equations may not have a finite solution. Consider a self-looping transition: $(q, x \lhd c, x := x - 1, q)$. We require $x \lhd c \in \mathcal{G}(q)$. Now, $\mathrm{wp}(x \lhd c, x := x - 1)$ is $x \lhd c + 1$ which should be in $\mathcal{G}(q)$ according to the above equation. Continuing this process, we need to add $x \lhd d$ for every natural number $d \geq c$. Indeed this is consistent with the undecidability of reachability when subtraction updates are allowed. We deal with the subject of finite solutions to the above equations later in this section. On the other hand, when the above system does have a solution with finite $\mathcal{G}(q)$ at every q, we can use the $\mathcal{A}$ simulation of Definition 5 and its approximation $\preccurlyeq^{LU}_{\mathcal{A}}$ to get an algorithm.

Proposition 2. *Let* $\mathcal{A} = (Q, X, q_0, T, F)$ *be a UTA. Let* $\{\mathcal{G}(q)\}_{q\in Q}$ *be the least solution to the equations given in Definition 9. Then, the relation* $\preccurlyeq_{\mathcal{A}}$ *is a simulation on the configurations of* $\mathcal{A}$*.*

Lemma 7. *For a UTA* $\mathcal{A}$*, assume that the least solution* $\{\mathcal{G}(q)\}_{q\in Q}$ *to the state-based guards equations is finite. Then the relation* $\preccurlyeq^{LU}_{\mathcal{A}}$ *is a finite simulation on the configurations of* $\mathcal{A}$*.*

Finite Solution to the State-Based Guards Equations. The least solution to the equations of Definition 9 can be obtained by a standard Kleene iteration for fixed points computation. For each $i \geq 0$ and each state q, define:

$$\mathcal{G}^0(q) = \bigcup_{(q,g,up,q')\in T} \{g\} \cup \mathrm{wp}(\sqsubseteq_{\{x\geq 0 | x\in X\}}, up)$$

$$\mathcal{G}^{i+1}(q) = \bigcup_{(q,g,up,q')\in T} \mathcal{G}^i(q) \cup \mathrm{wp}(\sqsubseteq_{\mathcal{G}^i(q')}, up)$$

The iteration stabilizes when there exists a k satisfying $\mathcal{G}^{k+1}(q) = \mathcal{G}^k(q)$ for all q. At stabilization, the values $\mathcal{G}^k(q)$ satisfy the equations of Definition 9, and give the required $\mathcal{G}(q)$. However, as we mentioned earlier, this iteration might not stabilize at any k. We will now develop some observations that will help detect after finitely many steps if the iteration will stabilize or not.

Suppose we colour the set $\mathcal{G}^{i+1}(q)$ to *red* if either there exists a diagonal constraint $x - y \lhd c \in \mathcal{G}^{i+1}(q) \setminus \mathcal{G}^i(q)$ (a new diagonal is added) or there exists a

non-diagonal constraint $x \triangleleft c$ or $c \triangleleft x$ in $\mathcal{G}^{i+1}(q) \setminus \mathcal{G}^i(q)$ such that the constant c is strictly bigger than c' for respectively every non-diagonal $x \triangleleft c'$ or $c' \triangleleft x$ in $\mathcal{G}^i(q)$ (a non-diagonal with a bigger constant is added). If this condition is not applicable, we colour the set $\mathcal{G}^{i+1}(q)$ *green*. The next observations say that the iteration terminates iff we reach a stage where all sets are green. Intuitively, once we reach green, the only constraints that can be added are non-diagonals having smaller (non-negative) constants and hence the procedure terminates.

Lemma 8. *Let $i > 0$. If $\mathcal{G}^i(q)$ is green for all q, then $\mathcal{G}^{i+1}(q)$ is green for all q.*

Lemma 9. *Let $K = 1 + |Q| \cdot |X| \cdot (|X| + 1)$. If there is a state p such that $\mathcal{G}^K(p)$ is red, then there is no i such that $\mathcal{G}^i(q)$ is green for all q.*

As to why the bound $K = 1 + |Q| \cdot |X| \cdot (|X| + 1)$ in the lemma above: a red state at stage i arises due to the addition of a constraint φ_i at state p_i, which in turn depends on a state p_{i-1} marked red at stage $i-1$ due to constraint φ_{i-1}. If we iterate sufficiently long, we will hit a state p, a sequence of transitions from p to p and a constraint φ such that computing the weakest precondition over this loop will give a new constraint with the same set of clocks as φ but with a different constant. This part can be iterated infinitely often.

Proposition 3. *The least solution of the local constraint equations for a UTA is finite iff $\mathcal{G}^K(q)$ is green for all q and where $K = 1 + |Q| \cdot |X| \cdot (|X| + 1)$.*

Theorem 4. *Let $\mathcal{A}$ be a UTA. It is decidable whether the equations in Definition 9 have a finite solution. When these equations do have a finite solution, zone graph enumeration using $\preccurlyeq_{\mathcal{A}}^{LU}$ is a sound, complete and terminating procedure for the reachability problem.*

All decidable classes of [12] can be shown decidable with our approach, by showing stabilization of the $\mathcal{G}(q)$ computation.

Lemma 10. *Reachability is decidable in UTA where: guards are non-diagonals and updates are of the form $x := c$, $x := y$, $x := y + c$ where $c \geq 0$ or, guards include diagonal constraints and updates are of the form $x := c$, $x := y$.*

6 Experiments

We have implemented the reachability algorithm for timed automata with diagonal constraints (and only resets as updates) based on the simulation approach (p. xx) using the $\preccurlyeq_{\mathcal{A}}^{LU}$ simulation (Definition 7) for pruning zones. The algorithm for $Z \sqsubseteq_{\mathcal{G}}^{LU} Z'$ comes from Sect. 4. Experiments are reported in Table 1. We take model *Cex* from [8,30] and *Fischer* from [30]. We are not aware of any other "standard" benchmarks containing diagonal constraints. In addition to these two models, we introduce a new benchmark. This is an extension of the job-shop scheduling using (diagonal-free) timed automata [1]. Here the tasks within a job were logically independent. We add some timing dependency between them

Table 1. Experiments: the column $\#\mathcal{D}$ gives the number of diagonal constraints. Four methods have been reported in the table. First two methods, TChecker with our simulation relation $\sqsubseteq_{\mathcal{G}}^{LU}$ and UPPAAL engine for diagonals, have been run on $\mathcal{A}$, the automata containing diagonal constraints. Whereas, the third and fourth methods are running diagonal-free engines of UPPAAL and TChecker on $\mathcal{A}_{df}$, a diagonal-free equivalent of $\mathcal{A}$. Experiments were run on macOS X with 2.3 GHz Intel core i5 processor, and 8 GB RAM. Time is reported in seconds. We set a timeout of 15 min.

Model	$\#\mathcal{D}$	$\mathcal{A}$: contains diagonals				$\mathcal{A}_{df}$: diagonal-free *equivalent* of $\mathcal{A}$			
		TChecker + $\sqsubseteq_{\mathcal{G}}^{LU}$		UPPAAL		UPPAAL		TChecker	
		Time	Nodes count	Time	Nodes count	Time	Nodes count	Time	Nodes count
Cex 2	4	0.047	241	0.026	2180	0.005	1039	0.067	1039
Cex 3	6	7.399	7111	111.168	182394	1.028	60982	40.092	60982
Cex 4	8	857.662	185209	Timeout	-	734.543	3447119	Timeout	-
Fischer 4	4	0.032	452	307.836	357687	0.009	1815	0.100	1815
Fischer 5	5	0.257	1842	Timeout	-	0.116	12511	1.856	12511
Fischer 7	7	15.032	26812	Timeout	-	174.560	693603	Timeout	-
Job Shop 3	12	0.420	278	23.093	31711	0.003	845	0.312	845
Job Shop 5	20	285.421	10592	Timeout	-	4.633	179607	150.811	179607

which gets naturally modeled using diagonal constraints. Each model considered above is a product of a number of k timed automata. In the table we write the name of the model and the number k of automata involved in the product. We also report the number of diagonal constraints in each of them.

Experimental Results. We report the results of four methods of handling diagonal constraints, as mentioned in the caption of Table 1. Under each method, we report on the number of zones enumerated and the time taken. The first method gives a huge gain over the second one (upto four orders of magnitude in the number of nodes, and even better for time) and gives a less marked, but still significant, gain over the third and fourth methods. We provide a brief explanation of this phenomenon. The performance of the reachability algorithm is dependent on three factors:

- parameters of extrapolation or simulation: M-simulations which use the maximum constant appearing in the guards, versus the LU-simulations which make a distinction between lower bound guards $c \lhd x$ and upper bound guards $x \lhd c$ (refer to [5] for the exact definitions of extrapolations based on these parameters, and [23] for simulations based on these parameters); LU-simulations are superior to M-simulations.
- computation of the parameters: global parameters which associate a bound to each clock versus the more local state based parameters as in Definition 4 which associate a set of bounds functions to each state [4]; local bounds are superior to global bounds.
- when diagonal constraints are present, whether zones get split or not: each time a zone gets split, new enumerations start from each of the new nodes; clearly, a no-splitting-of-zones approach is superior to zone splitting.

Algorithm of column 1 uses the superior heuristic in all the three optimizations above. The no-splitting-of-zones was possible thanks to our simulation approach, which temporarily splits zones for checking $Z \sqsubseteq_G^{LU} Z'$, but never starts a new exploration from any of the split nodes. The algorithm of column 2, which is implemented in the current version UPPAAL 4.1 uses the inferior heuristic in all the three above. In particular, it is not clear how the extrapolation approach can avoid the zone splitting in an efficient manner. The superiority of our approach gets amplified (by multiplicative factors) when we consider bigger products with many more diagonals. In the third and fourth methods, we give a diagonal free equivalent of the original model (c.f. Theorem 1) and use the UPPAAL and TChecker engines respectively, for diagonal free timed automata. The UPPAAL diagonal free engine is highly optimized, and makes use of the superior heuristics in the first two optimizations mentioned above (the third heuristic is not applicable now as it is a diagonal free automaton). The third and fourth methods can be considered as a good approximation of the zone splitting approach to diagonal constraints using LU-abstractions and local guards.

The second and the third methods are the only possibilities of verifying timed models coming with diagonal constraints in UPPAAL. Both these approaches are in principle prone to a $2^{\#\mathcal{D}}$ blowup compared to the first approach, where $\#\mathcal{D}$ gives the number of diagonal constraints. The table shows that a good extent of this blowup indeed happens. The UPPAAL diagonal free engine uses "minimal constraint systems" [6] for representing zones, whereas TChecker uses DBMs [15]. This explains why even with the same number of nodes visited, UPPAAL performs better in terms of time. We have not included in the table the comparison with two other works dealing with the same problem: the refined diagonal free conversion [30] and the extension of LU simulation for diagonals [18]. However, our results are better than the tables reported in these papers.

7 Conclusion

We have proposed a new algorithm for handling diagonal constraints in timed automata, and extended it to automata with general updates. Our approach is based on a simulation relation between zones. From our preliminary experiments, we can infer that the use of simulations is indispensable in the presence of diagonal constraints as zone-splitting can be avoided. Moreover, the fact that the simulation approach stores the actual zones (as opposed to abstracted zones in the extrapolation approach) has enabled optimizations for diagonal-free automata that work with dynamically changing simulation parameters (LU-bounds), which are learnt as and when the zones are expanded [22]. Working with actual zones is also convenient for finding cost-optimal paths in priced timed automata [11]. Investigating these in the presence of diagonal constraints is part of future work. Currently, we have not implemented our approach for updateable timed automata. This will also be part of our future work.

Working directly with a model containing diagonal constraints could be convenient (both during modeling, and during extraction of diagnostic traces) and can also potentially give a smaller automaton to begin with. We believe that our experiments provide hope that diagonal constraints can indeed be used.

References

1. Abdeddaim, Y., Asarin, E., Maler, O.: Scheduling with timed automata. Theor. Comput. Sci. **354**(2), 272–300 (2006). https://doi.org/10.1016/j.tcs.2005.11.018
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
3. Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., Yi, W.: TIMES b— a tool for modelling and implementation of embedded systems. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 460–464. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46002-0_32
4. Behrmann, G., Bouyer, P., Fleury, E., Larsen, K.G.: Static guard analysis in timed automata verification. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 254–270. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36577-X_18
5. Behrmann, G., Bouyer, P., Larsen, K.G., Pelánek, R.: Lower and upper bounds in zone-based abstractions of timed automata. STTT **8**(3), 204–215 (2006). https://doi.org/10.1007/s10009-005-0190-0
6. Bengtsson, J., Yi, W.: Timed automata: semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27755-2_3
7. Bérard, B., Petit, A., Diekert, V., Gastin, P.: Characterization of the expressive power of silent transitions in timed automata. Fundamenta Informaticae **36**(2,3), 145–182 (1998). https://doi.org/10.3233/FI-1998-36233
8. Bouyer, P.: Untameable timed automata!. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 620–631. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36494-3_54
9. Bouyer, P.: Forward analysis of updatable timed automata. Form. Methods Syst. Des. **24**(3), 281–320 (2004). https://doi.org/10.1023/B:FORM.0000026093.21513.31
10. Bouyer, P., Chevalier, F.: On conciseness of extensions of timed automata. J. Autom. Lang. Comb. **10**(4), 393–405 (2005). https://doi.org/10.25596/jalc-2005-393
11. Bouyer, P., Colange, M., Markey, N.: Symbolic optimal reachability in weighted timed automata. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 513–530. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_28
12. Bouyer, P., Dufourd, C., Fleury, E., Petit, A.: Updatable timed automata. Theor. Comput. Sci. **321**(2–3), 291–345 (2004). https://doi.org/10.1016/j.tcs.2004.04.003
13. Bouyer, P., Laroussinie, F., Reynier, P.-A.: Diagonal constraints in timed automata: forward analysis of timed systems. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 112–126. Springer, Heidelberg (2005). https://doi.org/10.1007/11603009_10
14. Daws, C., Tripakis, S.: Model checking of real-time reachability properties using abstractions. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 313–329. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054180

15. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52148-8_17
16. Ferrère, T.: The compound interest in relaxing punctuality. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (eds.) FM 2018. LNCS, vol. 10951, pp. 147–164. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95582-7_9
17. Fersman, E., Pettersson, P., Yi, W.: Timed automata with asynchronous processes: schedulability and decidability. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 67–82. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46002-0_6
18. Gastin, P., Mukherjee, S., Srivathsan, B.: Reachability in timed automata with diagonal constraints. In: Schewe, S., Zhang, L. (eds.) CONCUR 2018. Leibniz International Proceedings in Informatics (LIPIcs), vol. 118, pp. 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018). https://doi.org/10.4230/LIPIcs.CONCUR.2018.28
19. Gastin, P., Mukherjee, S., Srivathsan, B.: Fast algorithms for handling diagonal constraints in timed automata. CoRR abs/1904.08590 (2019). http://arxiv.org/abs/1904.08590
20. Hatvani, L., David, A., Seceleanu, C., Pettersson, P.: Adaptive task automata with earliest-deadline-first scheduling. In: Proceedings of the 14th International Workshop on Automated Verification of Critical Systems (AVoCS 2014), vol. 70. Electronic Communications of the EASST (2014). https://doi.org/10.14279/tuj.eceasst.70.975
21. Herbreteau, F., Point, G.: TChecker, April 2019 https://github.com/fredher/tchecker (v02)
22. Herbreteau, F., Srivathsan, B., Walukiewicz, I.: Lazy abstractions for timed automata. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 990–1005. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_71
23. Herbreteau, F., Srivathsan, B., Walukiewicz, I.: Better abstractions for timed automata. Inf. Comput. **251**, 67–90 (2016). https://doi.org/10.1016/j.ic.2016.07.004
24. Herbreteau, F., Tran, T.-T.: Improving search order for reachability testing in timed automata. In: Sankaranarayanan, S., Vicario, E. (eds.) FORMATS 2015. LNCS, vol. 9268, pp. 124–139. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22975-1_9
25. Ho, H.: Revisiting timed logics with automata modalities. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, pp. 67–76. ACM, New York (2019). https://doi.org/10.1145/3302504.3311818
26. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: LTSmin: high-performance language-independent model checking. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 692–707. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_61
27. Krčál, P., Yi, W.: Decidable and undecidable problems in schedulability analysis using timed automata. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 236–250. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24730-2_20
28. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. STTT **1**(1–2), 134–152 (1997). https://doi.org/10.1007/s100090050010

29. Ponge, J., Benatallah, B., Casati, F., Toumani, F.: Analysis and applications of timed service protocols. ACM Trans. Softw. Eng. Methodol. **19**(4), 11:1–11:38 (2010). https://doi.org/10.1145/1734229.1734230
30. Reynier, P.A.: Diagonal constraints handled efficiently in UPPAAL. In: Research report LSV-07-02. Laboratoire Spécification et Vérification, ENS Cachan, France (2007)
31. Wang, F.: Efficient verification of timed automata with BDD-like data structures. Int. J. Softw. Tools Technol. Transf. **6**(1), 77–97 (2004). https://doi.org/10.1007/s10009-003-0135-4
32. Yovine, S.: Kronos: a verification tool for real-time systems. (Kronos user's manual release 2.2). STTT **1**, 123–133 (1997). https://doi.org/10.1007/s100090050009
33. Zhao, J., Li, X., Zheng, G.: A quadratic-time dbm-based successor algorithm for checking timed automata. Inf. Process. Lett. **96**(3), 101–105 (2005). https://doi.org/10.1016/j.ipl.2005.05.027