Reachability for Updatable Timed Automata Made Faster and More Effective

Paul Gastin
LSV, ENS Paris-Saclay, CNRS, Université Paris–Saclay, France
paul.gastin@ens-paris-saclay.fr

Sayan Mukherjee
Chennai Mathematical Institute, India
sayanm@cmi.ac.in

B Srivathsan
Chennai Mathematical Institute, India
sri@cmi.ac.in

Abstract

Updatable timed automata (UTA) are extensions of classical timed automata that allow special updates to clock variables, like $x := x - 1$, $x := y + 2$, etc., on transitions. Reachability for UTA is undecidable in general. Various subclasses with decidable reachability have been studied. A generic approach to UTA reachability consists of two phases: first, a static analysis of the automaton is performed to compute a set of clock constraints at each state; in the second phase, reachable sets of configurations, called zones, are enumerated. In this work, we improve the algorithm for the static analysis. Compared to the existing algorithm, our method computes smaller sets of constraints and guarantees termination for more UTA, making reachability faster and more effective. As the main application, we get an alternate proof of decidability and a more efficient algorithm for timed automata with bounded subtraction, a class of UTA widely used for modelling scheduling problems. We have implemented our procedure in the tool TChecker and conducted experiments that validate the benefits of our approach.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models

Keywords and phrases Updatable timed automata, Reachability, Zones, Simulations, Static analysis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2020.47

Related Version A full version of the paper is available at `https://arxiv.org/abs/2009.13260`.

Funding Work supported by IRL ReLaX. Paul Gastin is supported by ANR project TickTac (ANR-18-CE40-0015). B Srivathsan is supported by CEFIPRA project IoTTTA (DST/CNRS ref. 2016-01). Sayan Mukherjee and B Srivathsan are additionally supported by Infosys Foundation (India).

1 Introduction

Timed automata [1] are finite automata equipped with real-time variables called clocks. Values of the clock variables increase at the same rate as time progresses. Transitions are guarded by constraints over the clock variables. During a transition, the value of a variable can be updated in several ways. In the classical model, variables can be reset to 0, written as a command $x := 0$ in transitions. Generalizations of this involve $x := c$ with $c \geq 0$ or $x := y + d$ where d is an arbitrary integer. Automata with these more general updates are called *Updatable Timed Automata (UTA)* [8, 6]. The updates provide a "discrete jump" facility during transitions. These are useful syntactic constructs for modeling real-time systems and have been used in several studies [12, 23, 19, 18, 25].

On the one hand, variables with both a continuous and a discrete flow offer modeling convenience. On the other hand, the discrete jumps are powerful enough to simulate counter machines through the use of $x := x + 1$ and $x := x - 1$ updates, in fact with zero time elapse during the entire simulation [8]. This makes reachability for this model undecidable. Various decidable subclasses have been investigated over the years [8, 12]. The most common technique to prove decidability involves showing the existence of a region automaton [1], which is a finite automaton accepting the (untimed) sequences of actions that have a timed run in the UTA. Although this gives decidability, the algorithm via the region construction is impractical due to the presence of exponentially many regions. Practical algorithms in current tools like UPPAAL [24], PAT [27], Theta [28] and TChecker [20] work with *zones*, which are bigger sets of configurations than regions and can be efficiently represented and manipulated using Difference-Bound Matrices (DBMs) [10]. Notably, these tools implement zone based algorithms only for UTA with restricted updates $x := c$ for $c \geq 0$, which behave similar to the reset $x := 0$. Most of the efforts in making the zone based algorithm more efficient have concentrated on this subclass of timed automata with only resets [4, 22, 26].

Recently, we have presented a zone based algorithm for updatable timed automata [14]. Due to the undecidability of the problem, it cannot deal with the whole class of UTA. It however covers the subclasses tabulated in [8]. The algorithm consists of two phases: first, a static analysis of the automaton is performed to compute a set of clock constraints at each state of the automaton; in the second phase, reachable sets of configurations, stored as zones, are enumerated. None of these phases has a guaranteed termination. If the static analysis terminates, a simulation relation between zones based on the constraints generated in the static analysis can be used to guarantee termination of the zone enumeration. Moreover, a smaller set of constraints in the static analysis gives a coarser simulation which leads to a faster zone enumeration. The simulation used in [14] lifts the efficient *LU*-simulation [4, 22] studied for diagonal-free reset-only timed automata to automata with diagonal constraints and updates.

Contributions. In this work, we strongly improve the static analysis of [14]. The new approach accumulates fewer clock constraints and terminates for a wider class of UTA. In particular, it terminates for *timed automata with bounded subtraction*, which was not the case before. This class contains updates $x := x - c$ with $c \geq 0$ along with resets. However, an update $x := x - c$ is allowed in a transition only when there is a promise that each configuration that can take this transition has a bounded x-value. This boundedness property gives decidability thanks to a finite region equivalence. This class has been used to model schedulability problems [12], where updates $x := x - c$ have been crucially used to model preemption. Thus, our new static analysis allows to use efficient simulations during the zone enumeration for this class.

At an algorithmic level, the new analysis is a slight modification of the older one. However, this makes some of the technical questions significantly harder: we show that deciding termination of the new analysis can be done in polynomial-time if the constants in the guards and updates are encoded in unary, whereas the problem is PSPACE-complete when the constants are encoded in binary. The older analysis does not depend on the encoding of constants, and has a polynomial-time algorithm for deciding termination.

For the experiments, the differences in the encoding and the hardness result do not carry much importance. The static analysis is implemented as a fixed-point iteration which can continue for a fixed number of steps determined by the size of the automaton, or can be stopped after a fixed time-out. We have implemented the new static analysis in the open source tool TChecker [20]. We noticed that the new method terminates and produces a result for more cases, and when both methods produce a result, the new method is faster.

Related work. Static analysis for timed automata without diagonal constraints and with updates restricted to $x := c$ and $x := y + c$ with $c \geq 0$ was studied in [3] in the context of M-simulations, which were implemented in earlier versions of UPPAAL and KRONOS [29]. Latest tools implement a more efficient LU-simulation [4, 22]. Our method clarifies how some optimizations of [3] can be lifted to the context of LU-simulations and more general updates, and also provides additional optimizations. TIMES [2] is a tool for modeling scheduling problems. It is mentioned in [12] that TIMES implements an algorithm using zones based on "the UPPAAL DBM library extended with a subtraction operator". However, the exact simulations used in the zone enumeration are not clear to us. A different approach to reachability is presented in [21] where the constraints needed for simulation are learnt during the zone enumeration directly. This potentially gives more relevant constraints and hence coarser simulations. On the flip side, it requires a sophisticated zone enumeration method with observable overheads. Moreover [21] deals with timed automata without diagonal constraints and general updates. Static analysis is lucrative since it is cheap, and maintains the reachability procedure as two simple steps. Apart from verification of UTA, studies on the expressive power of updates and diagonal constraints have been carried out in [8, 7]. Timed register automata [5] are a variant of UTA that have been looked at in the context of canonical representations.

Organization. Section 2 gives the preliminary definitions. Section 3 introduces the new static analysis approach. Some classes of UTA where the new static analysis performs better are discussed in Section 4. The subsequent Section 5 discusses the termination problem for the new static analysis. Section 6 provides the results of our experiments. We conclude with Section 7. The extended version [15] contains missing proofs and details about the models used for the experiments.

2 Preliminaries

We denote by $\mathbb{R}$ the set of reals, by $\mathbb{R}_{\geq 0}$ the non-negative reals, by $\mathbb{Z}$ the integers and by $\mathbb{N}$ the natural numbers. Let X be a finite set of variables over $\mathbb{R}_{\geq 0}$ called *clocks*. A *valuation* is a function $v\colon X \to \mathbb{R}_{\geq 0}$ that maps every clock to a non-negative real number. For $\delta \in \mathbb{R}_{\geq 0}$ we define valuation $v + \delta$ as $(v + \delta)(x) := v(x) + \delta$. The set of valuations is denoted by $\mathbb{V}$.

A *non-diagonal constraint* is an expression of the form $x \triangleleft c$ or $c \triangleleft x$, where $x \in X$, $c \in \mathbb{N}$ and $\triangleleft \in \{<, \leq\}$, that is, $x \triangleleft 3$ stands for either $x < 3$ or $x \leq 3$. A *diagonal constraint* is an expression of the form $x - y \triangleleft c$ or $c \triangleleft x - y$ where $x, y \in X$ are clocks and $c \in \mathbb{N}$. An *atomic constraint* is either a non-diagonal constraint or a diagonal constraint. We also consider two special atomic constraints $\top$ (true) and $\bot$ (false). A *constraint* φ is either an atomic constraint or a conjunction of atomic constraints, generated by the following grammar: $\varphi ::= \top \mid \bot \mid x \triangleleft c \mid c \triangleleft x \mid x - y \triangleleft c \mid c \triangleleft x - y \mid \varphi \wedge \varphi$ with $c \in \mathbb{N}$, $\triangleleft \in \{<, \leq\}$. Given a constraint φ and a valuation v, we write $v(\varphi)$ for the boolean expression that we get by replacing every clock x present in φ with the value $v(x)$. A valuation v is said to *satisfy* a constraint φ, written as $v \models \varphi$, if the expression $v(\varphi)$ evaluates to true. For every valuation v, we have $v \models \top$ and $v \not\models \bot$. Given a constraint φ, we define the set $[\![\varphi]\!] := \{v \mid v \models \varphi\}$.

An *update* $up\colon \mathbb{V} \mapsto \mathbb{V}$ is a partial function mapping valuations to valuations. The update up is specified by an *atomic update* for each clock x, given as either $x := c$ or $x := y + d$ where $c \in \mathbb{N}$, $d \in \mathbb{Z}$ and $y \in X$ (is possibly equal to x). We write up_x for the right hand side of the atomic update of x, that is, up_x is either c or $y + d$. 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 clock values are always non-negative. Given a valuation v and an update up, we define $v(up_x)$ to be c or $v(y) + d$ depending on up_x being c or $y + d$. We say $up(v) \geq 0$ if $v(up_x) \geq 0$ for all $x \in X$. In this case the *valuation* $up(v) \in \mathbb{V}$ is defined by $up(v)(x) = v(up_x)$ for all $x \in X$. In general, due to the presence of updates $up_x := y + d$ with $d < 0$, the update may not yield a clock valuation and for those valuations v, $up(v)$ is not defined. For example, if $v(x) = 5$ and $up_x = x - 10$ then $up(v)$ is undefined. Hence, the domain of the partial function $up\colon \mathbb{V} \to \mathbb{V}$ is the set of valuations v such that $up(v) \geq 0$. Updates can be used as transformations in timed automata transitions. An updatable timed automaton is an extension of a classical timed automaton which allows updates of clocks on transitions.

▶ **Definition 1.** *An* updatable timed automaton (UTA) $\mathcal{A} = (Q, X, q_0, T, F)$ *is given by a finite set Q of states, a finite set X of clocks, an initial state q_0, a set T of transitions and $F \subseteq Q$ of accepting states. Transitions are of the form (q, g, up, q') where g is a constraint (also called guard) and up is an update, $q, q' \in Q$ are the source and target states respectively.*

Fix a UTA $\mathcal{A} := (Q, X, q_0, T, F)$ for the rest of this section. A configuration of $\mathcal{A}$ is a pair (q, v) with $q \in Q$ and $v \in \mathbb{V}$. Semantics of $\mathcal{A}$ is given by a transition system over its configurations. There are two kinds of transitions: *delay* and *action*. For every configuration (q, v) and every $\delta \in \mathbb{R}_{\geq 0}$ there is a delay transition $(q, v) \xrightarrow{\delta} (q, v + \delta)$. For every transition $t := (q, g, up, q')$ in the automaton, there is an action transition $(q, v) \xrightarrow{t} (q', v')$ in the semantics if $v \models g$ (v satisfies the guard), $up(v) \geq 0$ (the update on v is defined) and $v' = up(v)$. The initial configuration is (q_0, v_0) with $v_0(x) = 0$ for every clock x. We write $(q, v) \xrightarrow{\delta, t} (q', v')$ for the sequence of delay δ and action t from (q, v). A *run* is an alternating sequence of delay and action transitions starting from the initial configuration: $(q_0, v_0) \xrightarrow{\delta_1, t_1} (q_1, v_1) \xrightarrow{\delta_2, t_2} \cdots \xrightarrow{\delta_n, t_n} (q_n, v_n)$. The run is accepting if $q_n \in F$.

The reachability problem for UTA asks if a given UTA has an accepting run. This problem is undecidable in general [8]. Various decidable fragments with a PSPACE-complete reachability procedure have been studied [8, 12, 14]. The basic reachability procedure involves computing sets of reachable configurations of the UTA stored as constraints which are popularly called as *zones* [9]. A zone is a set of valuations given by a conjunction of atomic constraints $x \triangleleft c$, $c \triangleleft x$, $x - y \triangleleft c$ and $c \triangleleft x - y$ with $c \in \mathbb{N}$ and $x, y \in X$. For example $(x - y \leq 5) \wedge (2 < x)$ is a zone. Given a state-zone pair (q, Z) (henceforth called a *node*) and a transition $t := (q, g, up, q')$, the set of valuations $Z_t := \{up(v) + \delta \mid v \in Z, v \models g, up(v) \geq 0, \delta \geq 0\}$ is a zone. This is the set of valuations obtained from the v in Z that satisfy the guard g of the transition, get updated to $up(v)$ and then undergo a delay δ. The initial node (q_0, Z_0) is obtained by delay from the initial configuration: $Z_0 := \{v_0 + \delta \mid \delta \geq 0\}$ is a zone. This lays the foundation for a reachability procedure: start with the initial node (q_0, Z_0); from each node (q, Z) that is freshly seen, explore the transitions $t := (q, g, up, q')$ out of q to compute resulting nodes (q', Z_t). If a pair (q, Z) with $q \in F$ is visited then the accepting state is reachable in the UTA. This naïve zone enumeration might not terminate [9]. For termination, *simulations* between zones are used.

A simulation relation on the UTA semantics is a preorder relation (in other words, a reflexive and transitive relation) $(q, v) \sqsubseteq (q, v')$ between configurations having the same state such that the relation is preserved (1) on delay: $(q, v + \delta) \sqsubseteq (q, v' + \delta)$ for all $\delta \in \mathbb{R}_{\geq 0}$ and (2) on actions: if $(q, v) \xrightarrow{t} (q_1, v_1)$, then $(q, v') \xrightarrow{t} (q_1, v_1')$ with $(q_1, v_1) \sqsubseteq (q_1, v_1')$ for all $t = (q, g, up, q_1)$. This relation gets naturally lifted to zones: $(q, Z) \sqsubseteq (q, Z')$ if for all $v \in Z$ there exists a $v' \in Z'$ such that $(q, v) \sqsubseteq (q, v')$. Intuitively, when $(q, Z) \sqsubseteq (q, Z')$, all sequences of transitions enabled from (q, Z) are enabled from (q, Z'). Therefore, all control states reachable from (q, Z) are reachable from (q, Z'). This allows for an optimization in the zone enumeration: a fresh node (q, Z) is not explored if there is an already visited node (q, Z') with $(q, Z) \sqsubseteq (q, Z')$. A simulation $\sqsubseteq$ is said to be *finite* if in every sequence of the form $(q, Z_0), (q, Z_1), \ldots$ there are two nodes (q, Z_i) and (q, Z_j) with $i < j$ such that $(q, Z_j) \sqsubseteq (q, Z_i)$. Using a finite simulation in the reachability procedure ensures termination. Various finite simulations have been studied in the literature, the most prominent being LU-simulation [4, 22, 13] and more recently the $\mathcal{G}$-simulation [14]. In addition to ensuring termination, one needs simulations which can quickly prune the search. One main focus of research in timed automata reachability has been in finding finite simulations which are efficient in pruning the search.

In a previous work [14], we introduced a new simulation relation for UTA, called the *$\mathcal{G}$-simulation*. This relation is parameterized by a set of constraints $\mathcal{G}(q)$ associated to every state q of the automaton. The sets $\mathcal{G}(q)$ are identified based on the transition sequences from q. We now present the basic definitions and properties of $\mathcal{G}$-simulation. The presentation differs from [14], but the essence of the technical content is the same.

▶ **Definition 2** (G-preorder)**.** *Given a finite or infinite set of constraints G, we say $v \sqsubseteq_G v'$ if for every $\delta \geq 0$, and every $\varphi \in G$: $v + \delta \models \varphi$ implies $v' + \delta \models \varphi$.*

We simply write $\sqsubseteq_\varphi$ instead of $\sqsubseteq_{\{\varphi\}}$ when $G = \{\varphi\}$ is a singleton set.

Directly from the definition of $\sqsubseteq_G$, we get that the relation $\sqsubseteq_G$ is a preorder. The definition also entails the following useful property: when $v \sqsubseteq_G v'$, $v \models \varphi$ implies $v' \models \varphi$ for all $\varphi \in G$. This is a first step towards getting a simulation on the UTA semantics. It says that all guards that v satisfies are satisfied by v', and hence all transitions enabled at v will be enabled at v' provided the transition guards are present in G. Valuations get updated on transitions and this property needs to be preserved over these updates. This motivates the following definition. It gives a constraint ψ such that $v \sqsubseteq_\psi v'$ will imply $up(v) \sqsubseteq_\varphi up(v')$.

▶ **Definition 3.** *Given an update up and a constraint φ, we define $up^{-1}(\varphi)$ to be the constraint resulting by simultaneously substituting up_x for x in φ: $up^{-1}(\varphi) := \varphi[up_x/x, \forall x \in X]$.*

For example, for $\varphi = x - y \triangleleft c$, $up^{-1}(x - y \triangleleft c) = up_x - up_y \triangleleft c$. Similarly, $up^{-1}(x \triangleleft c) = up_x \triangleleft c$ and $up^{-1}(c \triangleleft x) = c \triangleleft up_x$. Note that, $up^{-1}(\varphi)$ need not be in the syntax defined by the grammar for constraints. But, it can be easily rewritten to an equivalent constraint satisfying this syntax. For example: consider the constraint $x - y \triangleleft c$ and the update $up_x = z + d$ and $up_y = y$, then $up^{-1}(\varphi) = z + d - y \triangleleft c$, which is not syntactically a constraint. However, it is equivalent to the constraint $z - y \triangleleft c - d$. If $c - d < 0$, we further rewrite as $d - c \triangleleft y - z$. It is also useful to note that $up^{-1}(\varphi)$ may sometimes yield constraints equivalent to $\top$ or $\bot$. For example: if $\varphi = x \triangleleft c$ and $up_x = d$ with $d > c$, then the constraint $up^{-1}(\varphi)$ is equivalent to $\bot$, similarly, if $d < c$ then $up^{-1}(\varphi)$ is equivalent to $\top$.

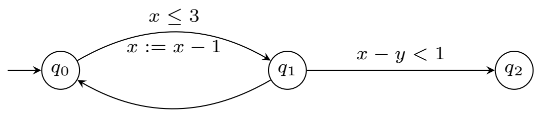
▶ **Lemma 4.** *Given a constraint φ, an update up and two valuations v, v' such that $up(v) \geq 0$ and $up(v') \geq 0$, if $v \sqsubseteq_{up^{-1}(\varphi)} v'$ then $up(v) \sqsubseteq_\varphi up(v')$.*

▶ **Definition 5** ($\mathcal{G}$-maps)**.** *Let $\mathcal{A} = (Q, X, q_0, T, F)$ be a UTA. A $\mathcal{G}$-map $\mathcal{G}_\mathcal{A}$ for UTA $\mathcal{A}$ is a tuple $(\mathcal{G}_\mathcal{A}(q))_{q \in Q}$ with each $\mathcal{G}_\mathcal{A}(q)$ being a set of atomic constraints, such that the following conditions are satisfied. For every transition $(q, g, up, q') \in T$:*

- *every atomic constraint of g belongs to $\mathcal{G}_\mathcal{A}(q)$,*
- *$up^{-1}(0 \leq x) \in \mathcal{G}_\mathcal{A}(q)$ for every $x \in X$,*
- *$up^{-1}(\varphi) \in \mathcal{G}_\mathcal{A}(q)$ for every $\varphi \in \mathcal{G}_\mathcal{A}(q')$ (henceforth called the* propagation criterion*)*

When the UTA $\mathcal{A}$ is clear from the context, we write $\mathcal{G}$ instead of $\mathcal{G}_\mathcal{A}$.

The propagation criterion allows to maintain the property described after Definition 2 even after the update occurring at transitions, and leads to a simulation relation on the configurations of the corresponding UTA, thanks to Lemma 4.



$$
\begin{array}{llllll}
\mathcal{G}(q_0) = \{x \leq 3,\ 1 \leq x\} & & \{\ \ldots,\ x - y < 2\} & & \{\ \ldots,\ x \leq 4,\ 2 \leq x\} & & \{\ \ldots,\ x - y < 3\} & \\
\mathcal{G}(q_1) = \{x - y < 1\} & \longrightarrow & \{\ \ldots,\ x \leq 3,\ 1 \leq x\} & \longrightarrow & \{\ \ldots,\ x - y < 2\} & \longrightarrow & \{\ \ldots,\ x \leq 4,\ 2 \leq x\} & \rightarrow \cdots \\
\mathcal{G}(q_2) = \{\} & & \{\} & & \{\} & & \{\} &
\end{array}
$$

Figure 1 Example automaton for which the $\mathcal{G}$-map computation of [14] does not terminate.

▶ **Definition 6** ($\mathcal{G}$-simulation)**.** *Given a $\mathcal{G}$-map $\mathcal{G}$, the relation $\sqsubseteq_{\mathcal{G}}$ on the UTA semantics defined as $(q, v) \sqsubseteq_{\mathcal{G}} (q', v')$ whenever $q = q'$ and $v \sqsubseteq_{\mathcal{G}(q)} v'$, is called the $\mathcal{G}$-simulation.*

In general, an automaton may not have *finite* $\mathcal{G}$-maps due to the propagation criterion generating more and more constraints. When a $\mathcal{G}$-map is finite, there is an algorithm to check $(q, Z) \sqsubseteq_{\mathcal{G}(q)} (q, Z')$. The fewer the constraints in a $\mathcal{G}(q)$, the larger is the simulation $\sqsubseteq_{\mathcal{G}(q)}$ (c.f. Definition 2). Hence there is more chance of getting $(q, Z) \sqsubseteq_{\mathcal{G}(q)} (q, Z')$ which in turn makes the enumeration more efficient. Moreover, fewer constraints in $\mathcal{G}(q)$ give a quicker simulation test $(q, Z) \sqsubseteq_{\mathcal{G}(q)} (q, Z')$. The goal therefore is to get a $\mathcal{G}$-map as small as possible. Notice that if $\mathcal{G}_1$ and $\mathcal{G}_2$ are $\mathcal{G}$-maps, then the map $\mathcal{G}_{min}$ defined as $\mathcal{G}_{min}(q) := \mathcal{G}_1(q) \cap \mathcal{G}_2(q)$ is also a $\mathcal{G}$-map. A static analysis of the automaton to get a $\mathcal{G}$-map is presented in [14]. The analysis performs an iterative fixed-point computation which gives the smallest $\mathcal{G}$-map (for the pointwise inclusion order) whenever it terminates. A procedure to detect if the fixed-point iteration will terminate at all is also given in [14].

3 A new static analysis with reduced propagation of constraints

In this section we give a refined propagation criterion, which cuts short certain propagations. We start with a motivating example. Figure 1 presents an automaton and illustrates the fixed-point iteration computing the smallest $\mathcal{G}$-map. Identity updates (like $y := y$) are not explicitly shown. Only the newly added constraints at each step are depicted. The first step adds constraints that meet the first two conditions of Definition 5. Note that $up^{-1}(0 \leq y)$ is $0 \leq y$ which is semantically equivalent to $\top$. So we do not add it explicitly to the $\mathcal{G}$-maps. Consider two transitions $(q_0, v) \xrightarrow{t} (q_1, up(v))$ and $(q_0, v') \xrightarrow{t} (q_1, up(v'))$ with $t = (q_0, x \leq 3, x := x-1, q_1)$, and up being $x := x-1$. Suppose we require $up(v) \sqsubseteq_{x-y<1} up(v')$. By Definition 2, we need to satisfy the condition: if $up(v) \models x-y < 1$, then $up(v') \models x-y < 1$. Rewriting in terms of v: if $v(x) - 1 - v(y) < 1$, then $v'(x) - 1 - v'(y) < 1$. In other words, we need: if $v \models x-y < 2$, then $v' \models x-y < 2$. This is achieved by adding $x-y < 2$, the constraint $up^{-1}(x - y < 1)$, to $\mathcal{G}(q_0)$ in the second step. This is the essence of the propagation criterion of Definition 5, which asks that for each $\varphi \in \mathcal{G}(q_1)$, we have $up^{-1}(\varphi) \in \mathcal{G}(q_0)$. The fixed-point computation iteratively ensures this criterion for each edge of the automaton. As illustrated, the computation does not terminate in Figure 1. There are three sources of increasing constants: (1) $x \leq 3$, $x \leq 4, \ldots$, (2) $1 \leq x$, $2 \leq x, \ldots$ and (3) $x - y < 1, x - y < 2, \ldots$.

We claim that this conservative propagation is unnecessary to get the required simulation. Suppose $v \sqsubseteq_{\mathcal{G}(q_0)} v'$ and $(q_0, v) \xrightarrow{t} (q_1, up(v))$, with $t := (q_0, x \leq 3, x := x - 1, q_1)$. Since t is enabled at v, we have $v(x) \leq 3$, hence $v'(x) \leq 3$ since guard $x \leq 3$ is present in $\mathcal{G}(q_0)$. We get $v, v' \models x - y \leq 3$ as $y \geq 0$ for all valuations. The presence of $x - y < 4, x - y < 5, \ldots$ at $\mathcal{G}(q_0)$ is useless as both v, v' already satisfy these guards. Stopping the propagation of $x - y < 3$ from $\mathcal{G}(q_1)$ will cut the infinite propagation due to (3). A similar reasoning cuts

Table 1 Cases where $up^{-1}(\varphi)$ can be eliminated or replaced by a constraint with a smaller constant. We write $\triangleleft$ and $\triangleleft_1$ to insist that the operator $\triangleleft$ need not be same as the operator $\triangleleft_1$.

	$up^{-1}(\varphi)$	g contains	$\text{pre}(\varphi, g, up)$
1.	$x \triangleleft d$	$x \triangleleft_1 c$	$\top$
2.	$d \triangleleft x$	$x \triangleleft_1 c$ with $c < d$	$c \leq x$
3.	$x - y \triangleleft d$ or $d \triangleleft x - y$	$x \triangleleft_1 c$ or $x - y \triangleleft_1 c$ or $e \triangleleft_1 x - y$ s.t. $c < d < e$	$\top$

the propagation of $x \leq 3$ from $\mathcal{G}(q_1)$ and stops (1). The remaining source (2) is trickier, but it can still be eliminated. Here is the main idea. Consider a constraint $3 \leq x \in \mathcal{G}(q_0)$ which propagates unchanged to $\mathcal{G}(q_1)$ and then back to $\mathcal{G}(q_0)$ as $up^{-1}(3 \leq x) = 4 \leq x$. This propagation can be cut since $v \sqsubseteq_{3 \leq x} v'$ already ensures $v \sqsubseteq_{4 \leq x} v'$ for the valuations that are *relevant*: the ones that satisfy the guard $x \leq 3$ of t. Indeed, $v, v' \models x \leq 3$ and $v \sqsubseteq_{3 \leq x} v'$ implies $v(x) \leq v'(x)$ which in turn implies $v \sqsubseteq_{4 \leq x} v'$. Overall, it can be shown that $\mathcal{G}(q_0) = \{x \leq 3,\ 3 \leq x,\ x - y < 2,\ x - y < 3\}$ and $\mathcal{G}(q_1) = \{x - y < 1\} \cup \mathcal{G}(q_0)$ suffices for the $\mathcal{G}$-simulation.

Taking guards into account for propagations. The propagation criterion of Definition 5 which is responsible for non-termination, is oblivious to the guard that is present in the transition. We will now present a new propagation criterion that takes the guard into account and cuts out certain irrelevant constraints. Consider a transition (q, g, up, q') and a constraint $\varphi \in \mathcal{G}(q')$. All we require is a constraint $\psi \in \mathcal{G}(q)$ such that $v \sqsubseteq_\psi v'$ *and* $v \models g$ implies $up(v) \sqsubseteq_\varphi up(v')$. The additional "and $v \models g$" was missing in the intuition behind the previous propagation. Of course, setting $\psi := up^{-1}(\varphi)$ is sufficient. However, the goal is to either eliminate the need for ψ or find a ψ with a smaller constant compared to $up^{-1}(\varphi)$. We will see that in many cases, we can even get the former, when we plug in the "and $v \models g$".

▶ **Definition 7** (pre of an atomic constraint φ under a "guard-update" pair (g, up))**.** *Let (g, up) be a pair of a guard and an update. For a constraint φ we define $\text{pre}(\varphi, g, up)$ to be an atomic constraint as given by Table 1, when g and $up^{-1}(\varphi)$ satisfy corresponding conditions. When the conditions of Table 1 do not apply, $\text{pre}(\varphi, g, up) = up^{-1}(\varphi)$.*

For a set of constraints $\mathcal{G}$, we define $\text{pre}(\mathcal{G}, g, up)$ to be the set $\bigcup_{\varphi \in \mathcal{G}} \{\text{pre}(\varphi, g, up)\}$.

Our aim is to replace the $up^{-1}(\varphi)$ in the older propagation criterion with $\text{pre}(\varphi, g, up)$. Before showing the correctness of this approach, we state a useful lemma that follows directly from the definition of $\mathcal{G}$-simulation.

▶ **Lemma 8.** *Let v, v' be valuations.*

- $v \sqsubseteq_{x \triangleleft d} v'$ *iff either* $v \not\models x \triangleleft d$ *or* $v'(x) \leq v(x)$
- $v \sqsubseteq_{d \triangleleft x} v'$ *iff either* $v' \models d \triangleleft x$ *or* $v(x) \leq v'(x)$

Readers familiar with the LU-simulation for diagonal-free automata [22] may recognize that the above lemma is almost an alternate formulation of the LU-simulation. The lemma makes a finer distinction between $<$ and $\leq$ in the constraints whereas LU does not.

The next proposition allows to replace the $up^{-1}(\varphi)$ in Definition 5 by $\text{pre}(\varphi, g, up)$ to get smaller sets of constraints at each q that still preserve the simulation. We write $v \sqsubseteq_g v'$ for $v \sqsubseteq_{C_g} v'$, where C_g is the set of atomic constraints in g.

▶ **Proposition 9.** *Let (g, up) be a guard-update pair, v, v' be valuations such that $v \models g$ and $v \sqsubseteq_g v'$, and φ be an atomic constraint. Then, $v \sqsubseteq_{\text{pre}(\varphi, g, up)} v'$ implies $v \sqsubseteq_{up^{-1}(\varphi)} v'$.*

Proof. When $\text{pre}(\varphi, g, up) = up^{-1}(\varphi)$, there is nothing to prove. We will now prove the theorem for the combinations given in Table 1.

(Case 1). From the hypothesis $v \sqsubseteq_g v'$, we get $v \sqsubseteq_{x \triangleleft_1 c} v'$. From the other hypothesis $v \models g$, we get $v \models x \triangleleft_1 c$. Therefore, by using the formulation of $v \sqsubseteq_{x \triangleleft_1 c} v'$ from Lemma 8, we get $v'(x) \leq v(x)$. This entails $v \sqsubseteq_{x \triangleleft d} v'$ for all upper bounded guards, once again from Lemma 8.

(Case 2). We have $\text{pre}(\varphi, g, up) = c \leq x$ and $c < d$. Moreover, as guard g contains $x \triangleleft_1 c$, we have $v'(x) \leq v(x)$ as in Case 1. Since v satisfies the guard, we get: $v'(x) \leq v(x) \leq c < d$. From Lemma 8, for such valuations, $v \sqsubseteq_{c \leq x} v'$ implies $v'(x) = v(x)$. Hence $v \sqsubseteq_{d \triangleleft x} v'$.

(Case 3). There are sub-cases depending on whether the guard contains a non-diagonal constraint or the diagonal constraints. When the guard contains $x \triangleleft_1 c$, we have $v'(x) \leq v(x) \leq c$ as above. Hence $v'(x - y) \leq c$ and $v(x - y) \leq c$. Since we are given that $c < d$, both v and v' satisfy the diagonal constraint $x - y \triangleleft d$ and neither of them satisfies $d \triangleleft x - y$. Notice that time elapse preserves the satisfaction of diagonal constraints as for every valuation u, $(u + d)(x - y) = u(x - y)$. From Definition 2, $v \sqsubseteq_\psi v'$ for a diagonal constraint ψ is satisfied if $v \not\models \psi$ or $v' \models \psi$. Hence, $v \sqsubseteq_{x-y \triangleleft d} v'$ and $v \sqsubseteq_{d \triangleleft x - y} v'$.

For the other sub-cases of the guard containing $x - y \triangleleft_1 c$ or $e \triangleleft_1 x - y$, the hypotheses $v \models g$, $v \sqsubseteq_g v'$ and the fact that $c < d < e$ ensure the same effect, that either v does not satisfy the diagonal constraint $up^{-1}(\varphi)$ or v' does. Hence, by definition $v \sqsubseteq_{up^{-1}(\varphi)} v'$. ◀

▶ **Definition 10** (Reduced $\mathcal{G}$-maps). *A $\mathcal{G}$-map is said to be* reduced *if for every transition (q, g, up, q'):*

- *every atomic constraint of g belongs to $\mathcal{G}(q)$,*
- $\text{pre}(0 \leq x, g, up) \in \mathcal{G}(q)$ *for every $x \in X$, and*
- $\text{pre}(\varphi, g, up) \in \mathcal{G}(q)$ *for every $\varphi \in \mathcal{G}(q')$ (reduced propagation)*

Recall the definition of $\mathcal{G}$-simulation of Definition 6. This is a relation $\sqsubseteq_\mathcal{G}$ defined as $(q, v) \sqsubseteq_\mathcal{G} (q', v')$ whenever $q = q'$ and $v \sqsubseteq_{\mathcal{G}(q)} v'$. The next theorem says that this relation stays a simulation even when the $\mathcal{G}$-map is reduced.

▶ **Theorem 11.** *Let $(\mathcal{G}(q))_{q \in Q}$ be a reduced $\mathcal{G}$-map. The relation $\sqsubseteq_\mathcal{G}$ is a simulation.*

As in the case of (non-reduced) $\mathcal{G}$-maps, notice that if $\mathcal{G}_1$ and $\mathcal{G}_2$ are reduced $\mathcal{G}$-maps, the map $\mathcal{G}_{min}$ given by $\mathcal{G}_{min}(q) = \mathcal{G}_1(q) \cap \mathcal{G}_2(q)$ is a reduced $\mathcal{G}$-map. There is therefore a smallest reduced $\mathcal{G}$-map, given by the pointwise intersection of all reduced $\mathcal{G}$-maps.

▶ **Lemma 12.** *The smallest reduced $\mathcal{G}$-map with respect to pointwise inclusion is the least fixed-point of the following system of equations:*

$$\mathcal{G}(q) = \bigcup_{(q,g,up,q')} \{\textit{atomic constraints of } g\} \cup \{\text{pre}(0 \leq x, g, up) \mid x \in X\} \cup \{\text{pre}(\varphi, g, up) \mid \varphi \in \mathcal{G}(q')\}$$

The smallest reduced $\mathcal{G}$-map can be computed by a standard Kleene iteration. For every state q and every $i \geq 0$:

$$\begin{aligned} \mathcal{G}^0(q) &= \bigcup_{(q,g,up,q')} \{\text{atomic constraints of } g\} \cup \{\text{pre}(0 \leq x, g, up) \mid x \in X\} \\ \mathcal{G}^{i+1}(q) &= \mathcal{G}^i(q) \cup \bigcup_{(q,g,up,q')} \{\text{pre}(\varphi, g, up) \mid \varphi \in \mathcal{G}^i(q')\} \end{aligned}$$

When $\mathcal{G}^{k+1} = \mathcal{G}^k$, a fixed-point has been found and $\mathcal{G}^k$ is a reduced map satisfying Definition 10. Moreover, $\mathcal{G}^k$ gives the least fixed-point to the system of equations of Lemma 12 and hence $\mathcal{G}^k$ is the smallest reduced $\mathcal{G}$-map. When $\mathcal{G}^{i+1} \neq \mathcal{G}^i$ for all i, the least fixed-point is infinite and no reduced $\mathcal{G}$-map for the automaton can be finite. For instance, if in the UTA of Figure 1, the guard $x \leq 3$ is removed, the smallest reduced $\mathcal{G}$-map will be infinite, and the fixed-point will continue forever, each iteration producing an $x - y < c$ with increasing constants c.

It is not clear apriori how to detect whether the fixed-point computation will terminate, or will continue forever. For the non-reduced $\mathcal{G}$-maps, [14] gives an algorithm that runs the fixed-point computation (using up^{-1} instead of pre) for a bounded number of steps and determines whether the computation will be non-terminating by looking for a certain witness. The reduced $\mathcal{G}$-map fixed-point is different due to Table 1, as certain propagations are disallowed (Cases 1 and 3), or truncated to a constant determined by the guard (Case 2). These optimizations are responsible for giving finite $\mathcal{G}$-maps even when the non-reduced $\mathcal{G}$-maps are infinite. This makes the termination analysis significantly more involved. We postpone this discussion to Section 5. In the next section, we identify some sufficient conditions that make the reduced $\mathcal{G}$-maps finite and describe how it leads to new applications. These observations throw more light on the mechanics of the reduced $\mathcal{G}$-computation and provide a preparation to the more technical Section 5.

4 Applications of the reduced propagation

We exhibit three subclasses of UTA for which the reduced $\mathcal{G}$-maps are superior than the non-reduced $\mathcal{G}$-maps: either reduced $\mathcal{G}$-maps are finite whereas non-reduced $\mathcal{G}$-maps are not guaranteed to be finite, or when both are finite, the reduced $\mathcal{G}$-map gives a bigger simulation.

Timed automata with bounded subtraction. Timed automata with diagonal constraints and updates restricted to classic resets $x := 0$ and subtractions $x := x - c$ with $c \geq 0$ have been used for modeling certain scheduling problems [12]. Reachability is undecidable for this restricted update model [8]. An important result in [12] is that reachability is decidable for a subclass called timed automata with *bounded subtraction*, and this decidability is used for answering the schedulability questions. Proof of decidability proceeds by constructing a region equivalence based on a maximum constant derived from the automaton. We prove that timed automata with bounded subtraction have finite reduced $\mathcal{G}$-maps. This gives an alternate proof of decidability and a zone-based algorithm using $\mathcal{G}$-simulation for this class of automata. This exercise also brings out the significance of reduced $\mathcal{G}$-maps: without the reduced computation, we cannot conclude finiteness.

▶ **Definition 13** (Timed Automata with Bounded Subtraction [12]). *A timed automaton with "subtraction" is an updatable timed automaton with updates restricted to the form $x := 0$ and $x := x - c$ for $c \geq 0$. Guards contain both diagonal and non-diagonal constraints.*

A timed automaton with "bounded subtraction" is a timed automaton with subtraction such that there is a constant M_x for each clock x satisfying the following property for all its reachable configurations (q, v): if there exists a transition (q, g, up, q') such that $v \models g$ and $up_x = x - c$ with $c > 0$, then $v(x) \leq M_x$.

It is shown in [12] that reachability is decidable for timed automata with bounded subtraction when the bounds M_x are known. This definition of bounded subtraction puts a semantic restriction over timed automata. Indeed, reachability is decidable only when the bounds M_x are apriori known. The following is a syntactically restricted class of timed automata, that captures the bounded subtraction model when the bounds M_x are given.

▶ **Definition 14** (Timed Automata with Syntactically Bounded Subtraction). *This is a timed automaton with subtraction such that, for every transition (q, g, up, q') and clock x, if $up_x = x - c$ with $c > 0$ then the guard g contains an upper bound constraint $x \lhd c'$ for some $c' \in \mathbb{N}$.*

▶ **Lemma 15.** *For every timed automaton with bounded subtraction $\mathcal{A}'$ where the bound M_x for every clock x is known, there exists a timed automaton with syntactically bounded subtraction $\mathcal{A}$ such that the runs of $\mathcal{A}$ and $\mathcal{A}'$ are the same.*

▶ **Theorem 16.** *The smallest reduced $\mathcal{G}$-maps are finite for timed automata with syntactically bounded subtraction.*

Proof. Let $\overline{M}$ be the maximum constant appearing among the guards and updates of the given automaton. Define $\overline{G}$ to be the (finite) set of all atomic constraints with constant at most $\overline{M}$. We will show that the *finite* map $\mathcal{G}$ assigning $\mathcal{G}(q) = \overline{G}$ for all q is a reduced $\mathcal{G}$-map. This then proves the theorem.

The first two conditions of Definition 10 are trivially true. It remains to show that $\text{pre}(\overline{G}, g, up) \subseteq \overline{G}$ for every transition (q, g, up, q'). Choose a constraint $\varphi \in \overline{G}$. Note that $\text{pre}(\varphi, g, up)$ is a constraint having a larger constant than φ only if up contains subtractions (since the other possible update is only a reset to 0 in this class). Thus, if up does not contain subtractions, from the construction of $\overline{G}$ it follows that $\text{pre}(\varphi, g, up) \subseteq \overline{G}$. Now, if $up_x = x - c$ for some clock x and $c > 0$, then g contains $x \lhd_1 c_1$ by definition. If $up^{-1}(\varphi)$ is some $x \lhd d$, then Case 1 of Table 1 gives $\text{pre}(\varphi, g, up) = \top$. If $up^{-1}(\varphi)$ is $d \lhd x$, from Case 2 of the table, we have $\text{pre}(\varphi, g, up) = c_1 \leq x$ or $\text{pre}(\varphi, g, up) = d \lhd x$ with $d \leq c_1$, which are both present in $\overline{G}$ by construction.

Finally, assume that $up^{-1}(\varphi)$ is a diagonal constraint $x - y \lhd d$ or $d \lhd x - y$ and Case 3 of Table 1 does not apply. We have $up_x = x - c_1$ with $c_1 \geq 0$ and $up_y = y - c_2$ with $c_2 \geq 0$ (a reset for x or y is not possible). Moreover, if $c_1 > 0$ (resp. $c_2 > 0$) then g contains some $x \lhd_1 c'_1$ (resp. $y \lhd_2 c'_2$). If $c_1 > 0$ then, since Case 3 does not apply, we get $d \leq c'_1 \leq \overline{M}$ and $up^{-1}(\varphi)$ belongs to $\overline{G}$. If $c_1 = 0$ and $c_2 > 0$ then the constraint φ is respectively $x - y \lhd d + c_2$ or $d + c_2 \lhd x - y$. Since $0 \leq d < d + c_2 \leq \overline{M}$, the constraint $up^{-1}(\varphi)$ is already in $\overline{G}$. ◀

Lemma 15 and Theorem 16 give an alternate proof of decidability and more importantly a zone based algorithm with optimized simulations for this model. The definition of timed automata with bounded subtraction can be seamlessly extended to include updates $x := y - c$ where $c \geq 0$ and x, y are potentially different clocks. Definition 14, Lemma 15 and Theorem 16 can suitably be modified to use $y \lhd c'$ instead of $x \lhd c'$. This preserves the decidability, with similar proofs, even for this extended class.

Clock bounded reachability. Inspired by Theorem 16, we consider the problem of clock-bounded reachability: given UTA and a bound $B \geq 0$, does there exist an accepting run $(q_0, v_0) \to (q_1, v_1) \to \cdots (q_n, v_n)$ where $v_i(x) \leq B$ for all i and all clocks x? This problem is decidable for the entire class of UTA. The algorithm starts with a modified zone enumeration: each new zone is intersected with $\bigwedge_x x \leq B$ before further exploration. This way, only the reachable configurations within the given bound are stored. The number of bounded zones is finite. Hence the enumeration will terminate without the use of any simulations. On the other hand, for efficiency, it is useful to prune the search through simulations. To use $\mathcal{G}$-simulation, we need a finite $\mathcal{G}$-map. Since we are interested in clock bounded reachability, we can inject the additional guard $\bigwedge_x x \leq B$ in all transitions. The following theorem says that for such automata, the reduced $\mathcal{G}$-map will be finite. This is not true with non-reduced $\mathcal{G}$-maps. For instance, consider a modification of the automaton in Figure 1 with all transitions having $x \leq 3 \land y \leq 3$. This does not help cutting any of the three sources of infinite propagation that have been discussed in the text below the figure.

▶ **Theorem 17.** *Suppose every transition of a UTA has a guard containing an upper constraint $x \triangleleft c$ for every clock. The reduced $\mathcal{G}$-map for such a UTA is finite.*

UTA with finite non-reduced $\mathcal{G}$-maps. Given a finite set of atomic constraints G, the algorithm for $Z \sqsubseteq_G Z'$ first divides G as $G^{nd} \cup G^d$ where G^{nd} and G^d are respectively the subsets of non-diagonal and diagonal constraints in G. From G^{nd}, two functions $L\colon X \mapsto \mathbb{N} \cup \{-\infty\}$ and $U\colon X \mapsto \mathbb{N} \cup \{-\infty\}$ are defined: $L(x) = \max\{c \mid c \triangleleft x \in G^{nd}\}$ and $U(x) = \max\{c \mid x \triangleleft c \in G^{nd}\}$. When there is no $c \triangleleft x$, $L(x) = -\infty$. Similarly for $U(x)$. Denote these functions as $L(G)$ and $U(G)$. Once G is rewritten as $L(G), U(G)$ and G^d, [14] gives a procedure to compute $Z \sqsubseteq_G Z'$.

For two $\mathcal{G}$-maps $\mathcal{G}_1$ and $\mathcal{G}_2$ we write $LU(\mathcal{G}_2) \leq LU(\mathcal{G}_1)$ if for every q and every clock x, $L(\mathcal{G}_2(q))(x) \leq L(\mathcal{G}_1(q))(x)$ and $U(\mathcal{G}_2(q))(x) \leq U(\mathcal{G}_1(q))(x)$. We write $\mathcal{G}_2^d \subseteq \mathcal{G}_1^d$ if $\mathcal{G}_2(q)^d \subseteq \mathcal{G}_1(q)^d$ for every q. It can be shown that for two $\mathcal{G}$-maps $\mathcal{G}_1$ and $\mathcal{G}_2$ with $LU(\mathcal{G}_2) \leq LU(\mathcal{G}_1)$ and $\mathcal{G}_2^d \subseteq \mathcal{G}_1^d$, the $\mathcal{G}_2$-simulation is bigger than the $\mathcal{G}_1$-simulation (using Lemma 8 for non-diagonals and the direct Definition 2 for diagonals). The following theorem asserts that when the non-reduced $\mathcal{G}$-map is finite, the reduced $\mathcal{G}$-map is finite and it induces a bigger simulation. The proof of this theorem proceeds by showing that every upper constraint $x \triangleleft c$ and diagonal constraint added by the reduced propagation is also added by the non-reduced propagation, and for every lower constraint $c \triangleleft x$ in the reduced $\mathcal{G}$, there is some $c' \triangleleft' x$ in the non-reduced $\mathcal{G}$ with $c \leq c'$.

▶ **Theorem 18.** *When the smallest (non-reduced) $\mathcal{G}$-map $\mathcal{G}_1$ is finite, the smallest reduced $\mathcal{G}$-map $\mathcal{G}_2$ is also finite. Moreover, $LU(\mathcal{G}_2) \leq LU(\mathcal{G}_1)$ and $\mathcal{G}_2^d \subseteq \mathcal{G}_1^d$.*

5 Termination of the reduced propagation

We present an algorithm and discuss the complexity for the problem of deciding whether the smallest reduced $\mathcal{G}$-map of a given automaton is finite. Briefly, we present a large enough bound B such that if the fixed point iteration does not terminate in B steps, it will never terminate and hence the smallest reduced $\mathcal{G}$-map given by the least fixed-point is infinite.

Let us first assume that there are no strict inequalities in the atomic constraints present in guards. For the termination analysis, we can convert all strict inequalities $<$ to weak inequalities $\leq$. The reduced propagation does not modify the nature of the inequality except in Case 2 of Table 1 where strict may change to weak. Any propagation in the original automaton is preserved in the converted automaton with the same constants and vice-versa. Hence the $\mathcal{G}$-map computation terminates in one iff it terminates in the other. We denote by c_φ the constant of an atomic constraint φ.

Let $\mathcal{A} = (Q, X, q_0, T, F)$ be some UTA. Let $M = \max\{c \mid c \text{ occurs in some guard of } \mathcal{A}\}$ and $L = \max\{|d| \mid d \text{ occurs in some update of } \mathcal{A}\}$. Let $\mathcal{G}$ be the smallest reduced $\mathcal{G}$-map computed by the least fixed-point of the equations in Lemma 12. We can show that this fixed-point computation does not terminate iff a constraint with a large constant is added to some $\mathcal{G}(q)$.

▶ **Proposition 19.** *The reduced $\mathcal{G}$-map computation does not terminate iff for some state q, there is an atomic constraint $\varphi \in \mathcal{G}(q)$ with a constant $c_\varphi > N = \max(M, L) + 2L|Q||X|^2$.*

For the analysis, we make use of strings of the form $x \leq$, $\leq x$, $x - y \leq$ and $\leq x - y$ where $x, y \in X$ and call them *contexts*. Given a context $\overline{\varphi}$ and a constant c, we denote by $\overline{\varphi}[c]$ the atomic constraint obtained by plugging the constant into the context.

In the proof, we shall use the notion of propagation sequence, which is a sequence $(q_i, \overline{\varphi}_i[c_i]) \to (q_{i+1}, \overline{\varphi}_{i+1}[c_{i+1}]) \to \cdots \to (q_j, \overline{\varphi}_j[c_j])$ such that for all $i \leq k < j$ we have $\overline{\varphi}_{k+1}[c_{k+1}] = \mathrm{pre}(\overline{\varphi}_k[c_k], g_k, up_k)$ for some transition $(q_{k+1}, g_k, up_k, q_k)$ of $\mathcal{A}$.

Proof of Proposition 19. The left to right implication of Proposition 19 is clear. Conversely, assume that $\overline{\varphi}[c] \in \mathcal{G}(q)$ for some $(q, \overline{\varphi}[c])$ with $c > \max(M, L) + 2L|Q||X|^2$. Consider the smallest $n \geq 0$ such that $\overline{\varphi}[c] \in \mathcal{G}^n(q)$. There is a propagation sequence $\pi = (q_0, \overline{\varphi}_0[c_0]) \to (q_1, \overline{\varphi}_1[c_1]) \to \cdots \to (q_n, \overline{\varphi}_n[c_n])$ such that $\overline{\varphi}_0[c_0] \in \mathcal{G}^0(q_0)$ and $(q_n, \overline{\varphi}_n[c_n]) = (q, \overline{\varphi}[c])$. Notice that $\overline{\varphi}_i[c_i] \in \mathcal{G}^i(q_i)$ for all $0 \leq i \leq n$. We first show that the propagation sequence π contains a positive cycle with large constants.

▶ **Lemma 20.** *We can find $0 < i < j \leq n$ such that $(q_i, \overline{\varphi}_i) = (q_j, \overline{\varphi}_j)$, $c_i < c_j$ and $\max(M, L) < c_k$ for all $i \leq k \leq j$.*

Proof. First, since $\overline{\varphi}_0[c_0] \in \mathcal{G}^0(q_0)$ we have $0 \leq c_0 \leq \max(M, L)$. We consider the last occurrence of a small constant in the propagation sequence. More precisely, we define $m = \max\{k \mid 0 \leq k < n \wedge c_k \leq \max(M, L)\}$. Hence, $c_k > \max(M, L)$ for all $m < k \leq n$.

Notice that, for $m < k < n$, the constraint in the sequence cannot switch from an upper diagonal to a lower diagonal and vice-versa. Indeed, assume that $\overline{\varphi}_k[c_k] = (x - y \leq c_k)$ and $\overline{\varphi}_{k+1}[c_{k+1}] = (c_{k+1} \leq y' - x')$. Then the update up_k contains $x := x' + d$, $y := y' - e$ with $c_{k+1} = d + e - c_k$. This is a contradiction with $d, e \leq L$ and $c_k, c_{k+1} > L$. Similarly, we can show that an upper (resp. lower) diagonal constraint cannot switch to a lower (resp. upper) non-diagonal constraint. On the other hand, it is possible to switch once from an upper (resp. lower) diagonal constraint to an upper (resp. lower) non-diagonal constraint.

The other remark is that $|c_{k+1} - c_k| \leq 2L$ for all $m \leq k < n$. Since $c_m \leq \max(M, L)$ and $c_n > \max(M, L) + 2L|Q||X|^2$, we find an increasing sequence $m < i_1 < i_2 < \cdots < i_\ell \leq n$ with $c_{i_1} < c_{i_2} < \cdots < c_{i_\ell}$ and $\ell > |Q||X|^2$. As noticed above, the $\overline{\varphi}_k$ are either all upper constraints or all lower constraints, hence the set $\{(q_k, \overline{\varphi}_k) \mid m < k \leq n\}$ contains at most $|Q||X|^2$ elements ($|X|$ for non-diagonals and $|X|(|X| - 1)$ for diagonals). Therefore, we find $i, j \in \{i_1, \ldots, i_\ell\}$ such that $i < j$ and $(q_i, \overline{\varphi}_i) = (q_j, \overline{\varphi}_j)$. Recall that $c_k > \max(M, L)$ for all $m < k \leq n$. ◀

The next step is to show that a positive cycle with large constants can be iterated resulting in larger and larger constants.

▶ **Lemma 21.** *Let $(q_i, \overline{\varphi}_i[c_i]) \to (q_{i+1}, \overline{\varphi}_{i+1}[c_{i+1}]) \to \cdots \to (q_j, \overline{\varphi}_j[c_j])$ be a propagation sequence with $(q_i, \overline{\varphi}_i) = (q_j, \overline{\varphi}_j)$, $\delta = c_j - c_i > 0$ and $M < c_k$ for all $i \leq k \leq j$. Then, $(q_i, \overline{\varphi}_i[c_i + \delta]) \to (q_{i+1}, \overline{\varphi}_{i+1}[c_{i+1} + \delta]) \to \cdots \to (q_j, \overline{\varphi}_j[c_j + \delta])$ is also a propagation sequence.*

This allows to conclude the proof of Proposition 19. Using Lemma 20 we obtain from π a positive cycle with large constants. This cycle can be iterated forever thanks to Lemma 21. We deduce that $\overline{\varphi}_i[c_i + k\delta] \in \mathcal{G}^i(q_i)$ for all $k \geq 0$ and the reduced $\mathcal{G}$-computation does not terminate. ◀

Algorithm to detect termination. Proposition 19 gives a termination mechanism: run the fixed-point computation $\mathcal{G}^0, \mathcal{G}^1, \ldots$, stop if either it stabilises with $\mathcal{G}^n = \mathcal{G}^{n+1}$ or if we add some constraint $\varphi \in \mathcal{G}^n(q)$ with $c_\varphi > N$. The number of pairs (q, φ) with $c_\varphi \leq N$ is $2N|Q||X|^2$ (the factor 2 is for upper or lower constraints). Therefore, the fixed-point computation stops after at most $2N|Q||X|^2$ steps and the total computation time is $\mathsf{poly}(M, L, |Q|, |X|)$. If the constants occurring in guards and updates of the UTA $\mathcal{A}$ are encoded in unary, the static analysis terminates in time $\mathsf{poly}(|\mathcal{A}|)$. If the constants are encoded in binary, (non-)termination of the $\mathcal{G}$-computation can be detected in NPSPACE = PSPACE: it suffices to search for a propagation sequence $(q_0, \varphi_0) \to (q_1, \varphi_1) \to \cdots \to (q_n, \varphi_n)$ such that $\varphi_0 \in \mathcal{G}^0(q_0)$ and $c_{\varphi_n} > N$. For this, we only need to store the current pair (q_k, φ_k), guess some transition $(q_{k+1}, g_k, up_k, q_k)$ of $\mathcal{A}$, and compute the next pair (q_{k+1}, φ_{k+1}) with $\varphi_{k+1} = \text{pre}(\varphi_k, g_k, up_k)$. This can be done with polynomial space. We can also show a matching PSPACE lower-bound.

Lower bound. We now show that when constants are encoded in binary, deciding termination of the reduced propagation is PSPACE-hard. To do this, we give a reduction from the control-state reachability of bounded one-counter automata.

A *bounded one-counter automaton* [17, 11] is given by (L, ℓ_0, Δ, b) where L is a finite set of states, ℓ_0 is an initial state, Δ is a set of transitions and $b \geq 0$ is the global bound for the counter. Each transition is of the form (ℓ, p, ℓ') where ℓ is the source and ℓ' the target state of the transition, $p \in [-b, +b]$ gives the update to the counter. A run of the counter automaton is a sequence $(\ell_0, c_0) \to (\ell_1, c_1) \to \cdots \to (\ell_n, c_n)$ such that $c_0 = 0$, each $c_i \in [0, b]$ and there are transitions $(\ell_i, p_i, \ell_{i+1})$ with $c_{i+1} = c_i + p_i$. All constants used in the automaton definition are encoded in binary. Reachability problem for this model asks if there exists a run starting from $(\ell_0, 0)$ to a given state ℓ_t with any counter value c_t. This problem is known to be PSPACE-complete [11]. We will now reduce the reachability for bounded one-counter automata to the problem of checking if the fixed-point computing the smallest reduced $\mathcal{G}$-map terminates (i.e, whether the smallest reduced $\mathcal{G}$-map is finite).

From a bounded one counter automaton $\mathcal{B} = (L, \ell_0, \Delta, b)$ we construct a UTA $\mathcal{A}_\mathcal{B}$. States of $\mathcal{A}_\mathcal{B}$ are $L \cup \{\ell'_0, \ell'_t\}$ where ℓ'_0 and ℓ'_t are new states not in L. There are two clocks x, y. For each transition (ℓ, p, ℓ') of $\mathcal{B}$, there is a transition (ℓ', g, up, ℓ) with guard $x \leq b \wedge y \leq 0$ and updates $x := x - p$ and $y := y$. We add some extra transitions using the new states ℓ'_0 and ℓ'_t: (1) $\ell_0 \xrightarrow{x-y\leq 0} \ell'_0$, (2) $\ell'_t \to \ell_t$ and (3) $\ell'_t \xrightarrow{x:=x, y:=y+1} \ell'_t$.

In the reduced $\mathcal{G}$-map computation for $\mathcal{A}_\mathcal{B}$, the constraint $x - y \leq 0$ is added to $\mathcal{G}^0(\ell_0)$. The propagation sequence starting from $(\ell_0, x - y \leq 0)$ mimicks the runs of the counter machine $\mathcal{B}$ with the value of the diagonal constraint $x - y \leq c$ giving the counter value. To keep this value bounded between 0 and b, we use Case 3 of Table 1. Guard $x \leq b$ disallows propagation of constraints $x - y \leq d$ with $d > b$. But, it can allow d to go smaller and smaller, and at one point the constant becomes negative and the constraint gets rewritten: $x - y \leq b, x - y \leq b - 1, \ldots, x - y \leq 0, 1 \leq y - x, 2 \leq y - x$, etc. The presence of the constraint $y \leq 0$ in the guard eliminates $1 \leq y - x, 2 \leq y - x$, etc. once again due to Case 3.

▶ **Lemma 22.** *For every run $(\ell_0, 0) \to (\ell_1, c_1) \to \cdots \to (\ell_n, c_n)$ in $\mathcal{B}$, there is a propagation sequence $(\ell_0, x - y \leq 0) \to (\ell_1, x - y \leq c_1) \to \cdots \to (\ell_n, x - y \leq c_n)$ in $\mathcal{A}_\mathcal{B}$.*

▶ **Lemma 23.** *For every propagation sequence $(\ell_0, x - y \leq 0) \to (\ell_1, x - y \leq c_1) \to \cdots \to (\ell_n, x - y \leq c_n)$ in $\mathcal{A}_\mathcal{B}$ with $\ell_i \in L$ for $0 \leq i \leq n$, there is a run $(\ell_0, 0) \to (\ell_1, c_1) \to \cdots \to (\ell_n, c_n)$ in $\mathcal{B}$.*

It now remains to notice that the only transition that can generate infinitely many constraints during the propagation is the loop $l'_t \to l'_t$, since the other transitions between states coming from the counter automaton have a guard to cut out infinite propagations. For this to happen some constraint needs to reach l_t, and propagate to l'_t via $l'_t \to l_t$.

▶ **Proposition 24.** *The final state is reachable in the counter automaton $\mathcal{B}$ iff the smallest reduced $\mathcal{G}$-map of $\mathcal{A}_\mathcal{B}$ is infinite.*

Table 2 # nodes is the number of nodes enumerated during a breadth-first-search; "-" denote that there was no answer for 20 minutes; N/A denotes not-applicable. Experiments were conducted on a MacBook Pro with 8 GB RAM, 2.3Ghz processor running macOS 10.14.

		New static analysis		Static analysis of [14]	
Model	Schedulable?	# nodes	time	# nodes	time
SporadicPeriodic-5	Yes	677	1.710s	-	-
SporadicPeriodic-20	No	852	1.742s	-	-
Mine-Pump	Yes	31352	7m 23.509s	-	-
Flower task triggering automaton: (computation time, deadline)					
(1,2), (1,2), (1,2)	No	212	0.057s	-	-
(1,10), (1,10), (1,10), (1,4)	Yes	105242	8m 57.256s	-	-
Worst-case task triggering automaton: (computation time, deadline)					
(1,2), (1,2), (1,2)	No	20	0.050s	-	-
(1,10), (1,10), (1,10), (1,4)	Yes	429	0.454s	-	-
12 copies of (1,20)	Yes	786	12m 5.250s	-	-
$\mathcal{A}_{gain} \times 3$	N/A	24389	7.611s	24389	12.402s
$\mathcal{A}_{gain} \times 4$	N/A	707281	14m 12.369s	707281	27m 13.540s

▶ **Theorem 25.** *Deciding termination of the reduced $\mathcal{G}$-map computation for a given UTA $\mathcal{A}$ is in* PTIME *if the constants in $\mathcal{A}$ are encoded in unary, and* PSPACE*-complete if the constants are encoded in binary.*

6 Experiments

We report on experiments conducted using the open source tool TChecker [20]. The models are given as networks of timed automata which communicate via synchronized actions. We have implemented the new static analysis discussed in Section 3. The older static analysis and zone enumeration with the $\mathcal{G}$-simulation were already implemented [14].

Compositionality of static analysis. Both these static analyses are performed individually on each component. For each local state q_i a map $\mathcal{G}(q_i)$ is computed. During the zone enumeration the product of the automata is computed on-the-fly. Each node is of the form (q, Z) where $q = \langle q_1, q_2, \dots, q_k\rangle$ is a tuple of local states, one from each component of the network and Z is a zone over all clocks of the network. The $\mathcal{G}$-map is then taken as $\mathcal{G}(q) = \bigcup_i \mathcal{G}(q_i)$. This approach creates a problem when there are shared clocks. A component i might update x and another component $j \neq i$ might contain a guard with x. The $\mathcal{G}$-maps computed component-wise will then not give a sound simulation. In our experiments, we construct models without shared clocks.

Benchmarks. Our primary benchmarks are models of task scheduling problems using the Earliest-Deadline-First (EDF) policy. Each task has a computation time and a deadline. Tasks are released either periodically or via a specification given as a timed automaton. The goal is to verify if for a given set of released tasks, all of them can be finished within their deadline. Preemption of tasks is allowed. This problem has been encoded as a reachability in a network of timed automata that uses bounded subtraction [12]. The main challenge is to model preemption. Each task t_i has an associated clock c_i which is reset as soon as the task starts to execute. While t_i is running, and some other task t_j preempts t_i, the clock c_i continues to elapse time. When t_j is done, an update $c_i := c_i - C_j$ is performed, where C_j denotes the computation time of t_j. This way, when t_i is scheduled again, clock c_i maintains the computation time that has elapsed since it was started. Whenever the EDF scheduler has to choose between task t_i and t_j, it chooses the one which is closest to its deadline. To get this, when t_i is released, a clock d_i is reset. Task t_i is prioritized over t_j if $D_i - d_i < D_j - d_j$ where D_i, D_j are the deadlines. We have constructed a model for the EDF scheduler based on these ideas (more details in [15]). For the experiments in Table 2, we consider some task release strategies given in the literature (SporadicPeriodic from TIMES tool, and a variant of Mine-Pump from [16]) and also create some of our own (Flower and Worst-case task triggers). The last model $\mathcal{A}_{gain}$ is an automaton with reset-to-zero only updates illustrating the gain when both static analyses terminate.

Comparison. For all the EDF examples, the old static analysis did not terminate, as seen in the last two columns of Table 2. This is expected since the model contains an update of the form $x := x - C$ which repeatedly adds guards $x \leq K, x \leq K + C, x \leq K + 2C, \ldots$. The new static analysis cuts this out, since the update $x := x - C$ occurs along with a guard $x \leq D$, making it a timed automaton with bounded subtraction. The $\mathcal{A}_{gain}$ example runs with both the static analyses. However, the new static analysis minimizes the propagation of diagonal constraints. The time taken by the simulation test used in the zone enumeration phase is highly sensitive to the number of diagonal constraints. Fewer diagonals therefore result in a faster zone enumeration. We have also tried our new static analysis for standard benchmarks of diagonal-free timed automata and observed no gain. In these models, the distance between a clock reset and a corresponding guard (in a component automaton) is small, usually within one or two transitions. Hence resets already cut out most of the guards and the optimizations of Table 1 do not seem to help here. We expect to gain primarily in the presence of updates or diagonal constraints. We also remark that the last experiment cannot be performed on the TIMES tool which is built for scheduling problems and the previous ones cannot be modeled in other timed automata tools UPPAAL, PAT and Theta since they cannot handle timed automata with subtraction updates. Our prototype therefore subsumes existing tools in terms of modeling capability.

7 Conclusion

We have presented a static analysis procedure for UTA. This method terminates for a wider class of UTA compared to [14], and hence makes powerful simulations applicable to these systems. We have experimented with a prototype implementation. At a technical level, we get a unifying framework to show decidability of the reachability problem for automata with diagonal constraints and updates $x := c$ and $x := y + d$ that covers the decidable subclasses of [8], [12] and [14], the only known decidable classes upto our knowledge. Our framework provides a high-level technique to extend to broader classes: to show decidability, check if there is a finite reduced $\mathcal{G}$-map (c.f. proof of Theorem 16 and the subsequent remark). Earlier route via regions [8, 12] requires a more involved low-level reasoning to show the correctness of the region equivalence. From a practical perspective, we have a prototype with a richer modeling language and a more efficient way to handle updates than the existing real-time model checkers. As future work, we plan to engineer the prototype to make it applicable for bigger models and release the implementation and benchmarks in the public domain.

References

1 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

2 Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. TIMES: A tool for schedulability analysis and code generation of real-time systems. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 2791 of *Lecture Notes in Computer Science*, pages 60–72. Springer, 2003.

3 Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim G. Larsen. Static guard analysis in timed automata verification. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–270. Springer, 2003.

4 Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *International Journal on Software Tools for Technology Transfer*, 8(3):204–215, 2006.

5 Mikolaj Bojanczyk and Slawomir Lasota. A machine-independent characterization of timed languages. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 7392 of *Lecture Notes in Computer Science*, pages 92–103. Springer, 2012.

6 Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.

7 Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 10(4):393–405, 2005.

8 Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.

9 Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.

10 David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.

11 John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is pspace-complete. *Inf. Comput.*, 243:26–36, 2015.

12 Elena Fersman, Pavel Krcál, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Inf. Comput.*, 205(8):1149–1172, 2007.

13 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in timed automata with diagonal constraints. In *International Conference on Concurrency Theory (CONCUR)*, volume 118 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

14 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. In *Computer Aided Verification (CAV)*, pages 41–59, Cham, 2019. Springer International Publishing.

15 Paul Gastin, Sayan Mukherjee, and B Srivathsan. Reachability for updatable timed automata made faster and more effective, 2020. `arXiv:2009.13260`.

16 Thorsten Gerdsmeier and Rachel Cardell-Oliver. Analysis of scheduling behaviour using generic timed automata. *Electronic Notes in Theoretical Computer Science*, 42:143–157, 2001. Computing: The Australasian Theory Symposium (CATS).

17 Christoph Haase, Joël Ouaknine, and James Worrell. Relating reachability problems in timed and counter automata. *Fundam. Inform.*, 143(3-4):317–338, 2016.

18 Leo Hatvani, Alexandre David, Cristina Cerschi Seceleanu, and Paul Pettersson. Adaptive task automata with earliest-deadline-first scheduling. *ECEASST*, 70, 2014.

19 Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998.

20 Frédéric Herbreteau and Gerald Point. TChecker. URL: `https://github.com/ticktac-project/tchecker`.

21 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Lazy abstractions for timed automata. In *Computer Aided Verification (CAV)*, volume 8044 of *Lecture Notes in Computer Science*, pages 990–1005. Springer, 2013.

22 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Information and Computation*, 251:67–90, 2016.

23 Pavel Krcál, Martin Stigge, and Wang Yi. Multi-processor schedulability analysis of preemptive real-time tasks with variable execution times. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 4763 of *Lecture Notes in Computer Science*, pages 274–289. Springer, 2007.

24 Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

25 Yuki Osada, Tim French, Mark Reynolds, and Harry Smallbone. Hourglass automata. In *Games, Automata, Logic and Formal verification (GandALF)*, volume 161 of *EPTCS*, pages 175–188, 2014.

26 Victor Roussanaly, Ocan Sankur, and Nicolas Markey. Abstraction refinement algorithms for timed automata. In *Computer Aided Verification (CAV)*, volume 11561 of *Lecture Notes in Computer Science*, pages 22–40. Springer, 2019.

27 Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. Pat: Towards flexible verification under fairness. In *Computer Aided Verification (CAV)*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer, 2009.

28 Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. Theta: a framework for abstraction refinement-based model checking. In Daryl Stewart and Georg Weissenbacher, editors, *Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 176–179, 2017. `doi:10.23919/FMCAD.2017.8102257`.

29 Sergio Yovine. Kronos: A verification tool for real-time systems. (Kronos user's manual release 2.2). *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997.