MITL Model Checking via Generalized Timed Automata and a New Liveness Algorithm

S. Akshay
Department of CSE, Indian Institute of Technology Bombay, Mumbai, India

Paul Gastin
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, 91190, Gif-sur-Yvette, France
CNRS, ReLaX, IRL 2000, Siruseri, India

R. Govind
Uppsala University, Sweden

B. Srivathsan
Chennai Mathematical Institute, India
CNRS, ReLaX, IRL 2000, Siruseri, India

Abstract

The translation of Metric Interval Temporal Logic (MITL) to timed automata is a topic that has been extensively studied. A key challenge here is the conversion of future modalities into equivalent automata. Typical conversions equip the automata with a guess-and-check mechanism to ascertain the truth of future modalities. Guess-and-check can be naturally implemented via alternation. However, since timed automata tools do not handle alternation, existing methods perform an additional step of converting the alternating timed automata into timed automata. This "de-alternation" step proceeds by an intricate finite abstraction of the space of configurations of the alternating automaton.

Recently, a model of generalized timed automata (GTA) has been proposed. The model comes with several powerful additional features, and yet, the best known zone-based reachability algorithms for timed automata have been extended to the GTA model, with the same complexity for all the zone operations. An attractive feature of GTAs is the presence of future clocks which act like timers that guess a time to an event and stay alive until a timeout. Future clocks seem to provide another natural way to implement the guess-and-check: start the future clock with a guessed time to an event and check its occurrence using a timeout. Indeed, using this feature, we provide a new concise translation from MITL to GTA. In particular, for the timed until modality, our translation offers an exponential improvement w.r.t. the state-of-the-art.

Thanks to this conversion, MITL model checking reduces to checking liveness for GTAs. However, no liveness algorithm is known for GTAs. Due to the presence of future clocks, there is no finite time-abstract bisimulation (region equivalence) for GTAs, whereas liveness algorithms for timed automata crucially rely on the presence of the finite region equivalence. As our second contribution, we provide a new zone-based algorithm for checking Büchi non-emptiness in GTAs, which circumvents this fundamental challenge.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models; Theory of computation → Quantitative automata; Theory of computation → Logic and verification

Keywords and phrases MITL model checking, timed automata, zones, liveness

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.5

Related Version *Full Version*: `https://arxiv.org/abs/2407.08452` [2]

1 Introduction

The translation of Linear Temporal Logic (LTL) [32] to Büchi automata is a fundamental problem in model checking, with a long history of theoretical advances [20, 36, 18], tool implementations [25, 14, 18, 30, 12] and practical applications [33, 34, 26, 27]. In the real-time setting, Metric Interval Temporal Logic (MITL) is close to LTL, with the modalities Next

(X) and Until (U) extended with timing intervals – for instance, $X_{[a,b]}p$ says that the next event is a p and it occurs within a delay $\theta \in [a, b]$. Model checking for MITL is known to be EXPSPACE-complete [4]. This has led to the study of "efficient" conversions from MITL to timed automata, with each new construction aiming to make the automata more succinct. Our work is another step in this direction.

There are two ways to interpret MITL formulae: over (continuous) timed signals [4, 29, 15] or (pointwise) timed words [6, 37, 11]. Since the current timed automata tools work with timed words, we stick with the pointwise semantics. The state-of-the-art for MITL-to-TA is based on an initial translation of MITL to one-clock Alternating Timed Automata (OCATA) [31]. It has been shown that these OCATA can be converted to a network of timed automata [9, 10]. The tool MightyL [11] implements the entire MITL-to-TA translation. One of the difficulties in the MITL-to-TA translation is the inherent mismatch between the logic and the automaton in the way timing constraints are enforced. A future modality declares that a certain event takes place at a certain timing distance, *in the future*. In a timed automaton, *clocks* measure time elapsed since some event *in the past* and check constraints on these values. To implement a future modality, the automaton needs to make a prediction about the event and verify that the prediction is indeed true. Therefore, each prediction typically resets a clock and stores a new obligation in the state. The automaton needs to discharge these obligations at the right times in the future.

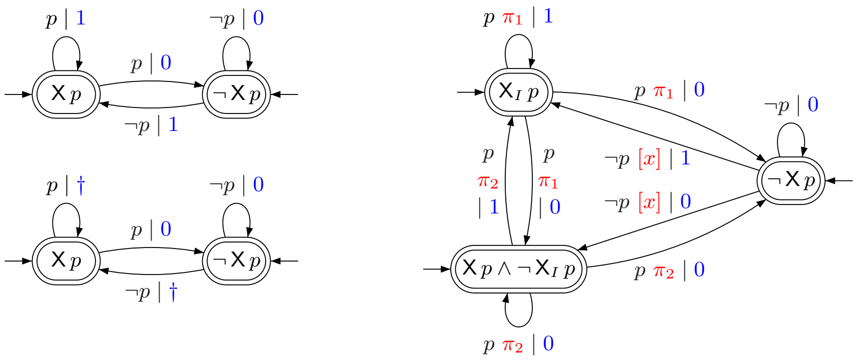

Figure 1 (top left) Büchi transducer (with outputs) for LTL formula $\mathsf{X}\,p$ (right) Timed transducer with clock x for MITL formula $\mathsf{X}_I\,p$; $\pi_1 := x \in I; [x]$, $\pi_2 := x \notin I; [x]$ (bottom left) a hypothetical transducer with a variable θ that predicts time to next action; $\dagger := \theta \in I\,?\,1 : 0$.

Figure 1 (top left) shows an automaton with outputs for the LTL formula $\mathsf{X}\,p$. On an infinite word $w_1 w_2 \ldots$ (where each w_i is a subset of atomic propositions) the automaton outputs 1 at w_i iff w_{i+1} contains p. While reading w_i, the automaton needs to guess whether $p \in w_{i+1}$ or not. Depending on the guess, it stores an appropriate obligation. This is reflected in the states and transitions: transitions with output 1 go to a state $\mathsf{X}\,p$ which can only read p next, whereas those with output 0 go to $\neg\mathsf{X}\,p$ which can only read $\neg p$. The $\mathsf{X}\,p$ and $\neg\mathsf{X}\,p$ can be seen as obligations that the automaton has to discharge from the state.

Now, let us consider a timed version $\mathsf{X}_I\,p$ interpreted on timed words. An automaton for $\mathsf{X}_I\,p$ needs to guess whether the next letter is a p *and* if so, whether it appears within θ time units for some $\theta \in I$. Figure 1 (bottom left) represents a hypothetical automaton that implements this idea: assuming it has access to a variable θ which contains the time to the next event, the output should depend on whether $\theta \in I$ or not. This is exactly what the if-then-else condition $\dagger$ does: if $\theta \in I$ output 1, else output 0. Classical timed automata do not have direct access to θ. They implement this idea differently, by making use of extra states. Figure 1 (right) shows a timed automaton for $\mathsf{X}_I\, p$. The state $\mathsf{X}\, p$ is split into two different obligations: $\mathsf{X}_I\, p$ where the timing constraint is satisfied, and $\mathsf{X}\, p \wedge \neg \mathsf{X}_I\, p$ where it is not. The outgoing guards discharge these obligations. This example shows the convenience of having access to a variable that can predict time to future events.

This is precisely where the recently proposed model of *Generalized Timed Automata* (GTA) [1] enters the picture. This model subsumes event-clock automata [5] and automata with timers [13]. GTA come equipped with the additional resources to implement predictions better. GTA have two types of clocks: *history clocks* and *future clocks*. History clocks are similar to the usual clocks of timed automata. Future clocks are like timers, but instead of starting them at some non-negative value and making them go down to zero, they get started with some arbitrary negative value and go up until they hit zero. For example, in Figure 1 (bottom left), each transition can start a future clock, guessing the time to the next event. This immediately gives us the required θ. The exact GTA for $\mathsf{X}_I\, p$ is quite close to Figure 1 (bottom left) and is given in Figure 4. Apart from the use of future clocks, the syntax of transitions in a GTA is much richer than a guard-reset pair as in timed automata. Transitions contain an "instantaneous timed program", which consists of a sequence of guards, resets and releases (for future clocks). When difference constraints are present, the model becomes powerful enough to encode counter machines and is therefore undecidable. A safe fragment, with a careful use of diagonal constraints is known to be decidable.

GTAs are advantageous in another sense. In spite of the powerful features, the best zone-based algorithms from the timed automata literature have been shown to suitably adapt to the GTA setting, with the same complexity for zone operations, and have been implemented in the tool TChecker [21]. Therefore, an MITL to GTA conversion allows us to capitalize on the features and succinct syntax of GTA, and at the same time, lets us model check MITL directly on richer GTA models. In summary:

- We provide a translation of MITL formulae to safe GTA. The translation is compositional and implementable, and yields an *exponential improvement* in the number of locations compared to the state-of-the-art technique for pointwise semantics [11], while the number of clocks remains the same up to a constant.
- Model checking MITL against GTA requires to solve the liveness problem for (safe) GTAs, which has been open so far. We settle the liveness question in this work. Zone based algorithms for event-clock automata have been studied in [19]. A notion of weak regions has been developed and this can be used for solving both reachability and liveness using zones. The GTA model that we consider in this paper strictly subsumes event-clock automata. In particular, the presence of diagonal constraints makes the problem more challenging. Our solution to liveness for GTAs therefore gives an alternate liveness procedure for event-clock automata, and also settles liveness for event-clock automata with diagonal constraints, a model defined in [8].

We remark that the techniques used in continuous semantics do not extend to pointwise semantics. In [15] the authors simplify general MITL formulae into formulae containing only one-sided intervals (of the form $[0, c]$ or $[c, \infty)$), for which automata are considerably simpler to construct. However, this simplification at the formula level works *only* in the continuous semantics – it does not work in the pointwise-semantics (as Lemma 4.3, 4.4 of [15] do not extend to pointwise-semantics). The fundamental difference is that in the continuous semantics we can assert a formula at any time point t. However, in pointwise-semantics, we can evaluate a formula only at *action points*, i.e., points where there is an actual action. For example, in continuous semantics one can rewrite $\mathsf{F}_{[a+c,b+c]}\, p$ as $\mathsf{F}_{[0,c]}\, \mathsf{G}_{[0,c]}\, \mathsf{F}_{[a,b]}\, p$ when $c \leq b - a$ (Lemma 4.3, [15]). However, in the pointwise semantics there may be no event in the interval $[0, c]$ on which we can evaluate $\mathsf{G}_{[0,c]}\, \mathsf{F}_{[a,b]}\, p$. Therefore, we need a completely different approach to deal with intervals in the pointwise semantics.

Organization of the paper. We start with preliminary definitions of Generalized Timed Automata (Section 2) and provide our solution to the liveness problem in Section 3. We present our MITL to GTA translation in Section 4. Missing proofs and additional explanations can be found in the full version available at [2].

2 Preliminaries

Let $X = X_F \uplus X_H$ be a set of real-valued variables called *clocks*, which is further partitioned into *future clocks* X_F and *history clocks* X_H. Let $\Phi(X)$ denote a set of *clock constraints* generated by the grammar: $\varphi ::= x - y \triangleleft c \mid \varphi \wedge \varphi$ where $x, y \in X \cup \{0\}$, $\triangleleft \in \{<, \leq\}$ and $c \in \overline{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, +\infty\}$ (the set of integers equipped with the two special values to say that a clock is "undefined"). We also allow *renamings* of clocks. Let perm_X be the set of permutations σ over $X \cup \{0\}$ mapping history (resp. future) clocks to history (resp. future) clocks ($\sigma(X_F) = X_F$ and $\sigma(X_H) = X_H$).

GTA syntax. A *Generalized Timed Automaton (GTA)* is given by $(Q, \Sigma, X, \Delta, \mathcal{I}, Q_f)$ where Q is a finite set of states, Σ is a finite alphabet of actions, $X = X_F \uplus X_H$ is a set of clocks partitioned into future clocks X_F and history clocks X_H. The initialization condition $\mathcal{I}$ is a set of pairs (q_0, g_0) where a pair consists of an initial state $q_0 \in Q$ and an initial guard $g_0 \in \Phi(X)$, and the accepting condition is given by a set $Q_f \subseteq Q$ of Büchi states. The transition relation $\Delta \subseteq (Q \times \Sigma \times \text{Programs} \times Q)$ contains transitions of the form $(q, a, \mathsf{prog}, q')$, where q is the source state, q' is the target state, a is the action triggering the transition, and prog is an *instantaneous timed program* generated by the grammar:

$$\mathsf{prog} := \mathsf{guard} \mid \mathsf{change} \mid \mathsf{rename} \mid \mathsf{prog}; \mathsf{prog}$$

where $\mathsf{guard} = g \in \Phi(X)$, $\mathsf{change} = [R]$ for an $R \subseteq X$, and $\mathsf{rename} = [\sigma]$ for a $\sigma \in \mathsf{perm}_X$.

Figure 4 with the blue parts removed illustrates a GTA. Both states ℓ_1 and ℓ_2 are initial, denoted by incoming arrows to each of them, and accepting, marked by the double circle. The initial guard is the trivial *true* constraint. The alphabet $\Sigma = \{0, 1\}$ (written in black). The constraint $-x \in I$ is short form for a conjunction of constraints requiring the clock to be in the interval I. For example, if $I = (4, 5]$, then $-x \in I$ is the constraint $4 < -x \wedge -x \leq 5$. During our MITL to GTA translation, we extend GTAs to include outputs (a formal definition is given in [2]). The dagger condition $(-x \in I) ? 1 : 0$ is a short form for two transitions, one which checks $-x \in I$ and outputs 1, and the other which checks $-x \notin I$ and outputs 0.

GTA semantics. A valuation of clocks is a function $v \colon X \cup \{0\} \mapsto \overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ which maps the special clock 0 to 0, history clocks to $\mathbb{R}_{\geq 0} \cup \{+\infty\}$ and future clocks to $\mathbb{R}_{\leq 0} \cup \{-\infty\}$. We denote by $\mathbb{V}(X)$ or simply by $\mathbb{V}$ the set of valuations over X. For a valuation $v \in \mathbb{V}$, define[1] $v \models y - x \triangleleft c$ as $v(y) - v(x) \triangleleft c$. We say that v *satisfies* a constraint φ, denoted as $v \models \varphi$, when v satisfies all the atomic constraints in φ. We denote by $v + \delta$ the *valuation* obtained from valuation v by increasing by $\delta \in \mathbb{R}_{\geq 0}$ the value of all clocks in X. Note that, from a given valuation, not all time elapses result in valuations since future clocks need to stay at most 0. We now define the *change* operation that combines the *reset* operation for history clocks (which sets history clocks to 0) and *release* operation for future clocks (which

[1] To allow evaluation of all the constraints in $\Phi(X)$, the addition and the unary minus operation on real numbers is extended [1] with the following conventions (i)$(+\infty) + \alpha = \alpha + (+\infty) = +\infty$ for all $\alpha \in \overline{\mathbb{R}}$, (ii) $(-\infty) + \beta = \beta + (-\infty) = -\infty$, as long as $\beta \neq +\infty$, and (iii) $-(+\infty) = -\infty$ and $-(-\infty) = +\infty$.

assigns a non-deterministic value to a future clock). Given a set of clocks $R \subseteq X$, we define $R_F = R \cap X_F$ as the set of future clocks in R, and $R_H = R \cap X_H$ as the set of history clocks in R. Then, $[R]v := \{v' \in \mathbb{V} \mid v'(x) = 0 \ \forall\, x \in R_H \text{ and } v'(x) = v(x) \ \forall\, x \notin R\}$. Observe that v' has no constraints for the future clocks in R, as they can take any arbitrary value in v'. For a valuation $v \in \mathbb{V}(X)$, and $\sigma \in \mathsf{perm}_X$, we define $[\sigma]v$ as $v \circ \sigma$, i.e., $([\sigma]v)(x) = v(\sigma(x))$ for all $x \in X \cup \{0\}$.

For valuations v, v' and a guard $g \in \Phi(X)$ we write $v \xrightarrow{g} v'$ when $v' = v \models g$, and $v \xrightarrow{[R]} v'$ when $R \subseteq X$ and $v' \in [R]v$, and $v \xrightarrow{[\sigma]} v'$ when $\sigma \in \mathsf{perm}_X$ and $v' = [\sigma]v$. When $\mathsf{prog} = \mathsf{prog}_1; \dots; \mathsf{prog}_n$, we write $v \xrightarrow{\mathsf{prog}} v'$ when there are valuations $v_1, \dots, v_n$ such that $v \xrightarrow{\mathsf{prog}_1} v_1 \xrightarrow{\mathsf{prog}_2} \cdots \xrightarrow{\mathsf{prog}_n} v_n = v'$. The semantics of the GTA $\mathcal{A}$ defined above is given by a transition system $\mathbb{TS}_{\mathcal{T}}$ whose states are *configurations* (q, v) of $\mathcal{A}$, where $q \in Q$ and $v \in \mathbb{V}$ is a valuation. A configuration (q, v) is initial if $v \models g$ for some $(q, g) \in \mathcal{I}$, and it is accepting if $q \in Q_f$. Transitions of $\mathbb{TS}_{\mathcal{T}}$ are of two forms: (1) *delay transition*: $(q, v) \xrightarrow{\delta} (q, v + \delta)$ if $v + \delta$ is a valuation, i.e., $(v + \delta) \models X_F \le 0$, and (2) *discrete transition*: $(q, v) \xrightarrow{t} (q', v')$ if $t = (q, a, \mathsf{prog}, q') \in \Delta$ and $v \xrightarrow{\mathsf{prog}} v'$. A *finite (respectively infinite) run* ρ of a GTA is a finite (respectively infinite) sequence of transitions from an initial configuration of $\mathbb{TS}_{\mathcal{A}}$: $(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \cdots$.

For example, consider the run of GTA in Figure 4 on a timed word $(1,1)(0,2)(1,3)(0,4)\dots$ (1 occurs at all odd numbers, and 0 at all even numbers, starting from first timestamp 1). The program $(x = 0); [x]$ used in the transitions first checks if x is 0, and then releases it to an arbitrary non-deterministic value. The run on the above word would be: $(\ell_1, x = -1) \xrightarrow{1, t_1} (\ell_2, x = -1) \xrightarrow{1, t_2} (\ell_1, x = -1) \cdots$. A transition $\xrightarrow{\delta, t}$ denotes a time elapse of δ followed by application of the program associated to transition t. At each point the value of x is released to -1, and is checked with the guard $x = 0$ at the next event.

An infinite run is accepting if it visits accepting configurations infinitely often. The run is said to be *Zeno* if $\Sigma_{i \ge 0} \delta_i$ is bounded and *non-Zeno* otherwise. In this work, we will be interested in *strongly non-Zeno* GTA: these are GTA where every accepting run is non-Zeno. It is possible to convert every GTA into a strongly non-Zeno GTA using a standard construction from timed automata literature [35]. In the rest of the document, we will drop the "strongly non-Zeno" prefix and simply say GTA.

Liveness problem. The *non-emptiness or liveness problem* for a GTA asks whether the given GTA has an accepting non-Zeno run. Due to our assumption about strong non-Zenoness, the question reduces to asking if a given GTA has an accepting run. Unfortunately, the non-emptiness problem even for finite words turns out to be undecidable for general GTA [1]. Therefore, we focus our attention on a restricted sub-class of GTA's for which non-emptiness in the finite words case is decidable, called safe GTA [1].

▶ **Definition 1** (Safe GTA [1]). *Given a GTA $\mathcal{A}$, let $X_D \subseteq X_F$ be the subset of future clocks used in diagonal guards of $\mathcal{A}$ between future clocks, i.e., if $x - y \triangleleft c$ with $x, y \in X_F$ occurs in some guard of $\mathcal{A}$ then $x, y \in X_D$. Then, a program* prog *is X_D-safe if clocks in X_D are checked for being 0 or $-\infty$ before being released and renamings $[\sigma]$ used in* prog *preserve X_D clocks ($\sigma(X_D) = X_D$). A GTA $\mathcal{A}$ is safe if it only uses X_D-safe programs on its transitions and the initial guard g_0 sets each history clock to either 0 or ∞.*

The GTA in Figure 4 is vacuously safe, since there are no diagonal constraints at all.

▶ Remark 2. Renaming operations may be considered as syntactic sugar allowing for more concise representations of GTAs. Indeed, we can transform a GTA $\mathcal{A}$ with renamings to an equivalent GTA $\mathcal{A}'$ without renamings by adding to the state the current permutation of clocks (composition of the permutations applied since the initial state) and change the programs of outgoing transitions accordingly. The number of states is multiplied by the number of permutations that may occur as described above.

Zones, zone graph and simulations. Reachability for GTA proceeds by an enumeration of its reachable configurations stored as constraint systems called *zones*. A zone over a set of variables $X \cup \{0\}$ is a conjunction of difference constraints $x - y \triangleleft c$ where $x, y \in X \cup \{0\}$, $\triangleleft \in \{<, \leq\}$ and $c \in \overline{\mathbb{Z}}$. For a zone Z and a valuation v, we write $v \in Z$ if the valuation v satisfies every constraint in Z. Therefore, we also interpret Z as a set of valuations, which satisfy its constraints.

A pair (q, Z) with q a control state and Z a zone represents $\{(q, v) \mid v \in Z\}$. Successors for (q, Z) can be defined based on the outgoing transitions of q. For a transition $t := (q, a, \mathsf{prog}, q')$, we write $(q, Z) \xrightarrow{t} (q', Z_t)$ if $Z_t = \{v' \mid v'$ is a valuation and $(q, v) \xrightarrow{t}\xrightarrow{\delta} (q', v')$ for some $v \in Z$ and $\delta \in \mathbb{R}_{\geq 0}\}$. It was shown in [1] that the successor Z_t is also a zone. This observation is used to define the notion of the *Zone graph* of a GTA.

▶ **Definition 3** (GTA zone graph [1]). *Given a GTA $\mathcal{A}$, its* GTA zone graph, *denoted* $\mathsf{GZG}(\mathcal{A})$, *is defined as follows: Nodes are of the form (q, Z) where q is a state and Z is a GTA zone. Initial nodes are pairs $(q_0, \overrightarrow{Z_0})$ where $(q_0, g_0) \in \mathcal{I}$ is an initial condition and Z_0 is given by $g_0 \wedge (X_F \leq 0) \wedge (X_H \geq 0)$ (Z_0 is the set of all valuations which satisfy the* initial constraint *g_0). For every node (q, Z) and every transition $t := (q, a, \mathsf{prog}, q')$ there is an edge $(q, Z) \xrightarrow{t} (q', Z_t)$ in the GTA zone graph.*

Finally, just as is the case for zone graphs for timed automata, $\mathsf{GZG}(\mathcal{A})$ is not guaranteed to be finite. In order to use it to check Büchi non-emptiness or reachability, we need a finite abstraction of the zone graph. The standard technique to obtain such finite abstractions is using the notion of *simulations*, that we recall next.

▶ **Definition 4** (Simulation). *A (time-abstract) simulation relation on the semantics of a GTA is a reflexive, transitive relation $(q, v) \preceq (q, v')$ relating configurations with the same control state and*

1. *for every $\delta \in \mathbb{R}_{\geq 0}$ such that $v + \delta \in \mathbb{V}$ is a valuation, there exists $\delta' \in \mathbb{R}_{\geq 0}$ such that $v' + \delta' \in \mathbb{V}$ is a valuation and $(q, v + \delta) \preceq (q, v' + \delta')$,*
2. *for every transition t, if $(q, v) \xrightarrow{t} (q_1, v_1)$ for some valuation v_1, then $(q, v') \xrightarrow{t} (q_1, v_1')$ for some valuation v_1' with $(q_1, v_1) \preceq (q_1, v_1')$,*
3. *for all future clocks $x \in X_F$, if $v(x) = -\infty$ then $v'(x) = -\infty$.*

For two GTA zones Z, Z', we say $(q, Z) \preceq (q, Z')$ if for every $v \in Z$ there exists $v' \in Z'$ such that $(q, v) \preceq (q, v')$.

3 Liveness for GTA

In this section, we will discuss a zone-based procedure to check liveness for safe generalized timed automata. We start by explaining how the standard zone based algorithm for solving liveness in classical timed automata can be adapted to the setting of safe GTAs. The approach for timed automata crucially depends on the existence of a finite time-abstract bisimulation between valuations, namely the region-equivalence [3]. However, there exists no such finite time-abstract bisimulation for GTAs (extension of a result of [19]), as illustrated in Figure 2. The issue is that we cannot forget (abstract) the values of future clocks, unlike history clocks where values above a maximum constant are equivalent. Therefore, our approach involves a significant deviation from the standard one.

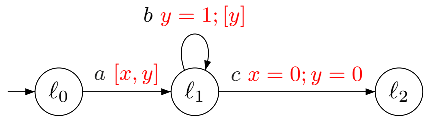

Figure 2 Example to illustrate no finite bisimulation in GTA; x is a future clock, y a history clock. The initial transition releases clock x to an arbitrary value, and resets y to 0. From configuration $\langle \ell_1, x = -n, y = 0\rangle$ ($n \in \mathbb{N}$), the only way to reach ℓ_2 is by executing $b^n c$, with 1 time unit between consecutive b's. Therefore, $\langle \ell_1, x = -n, y = 0\rangle$ and $\langle \ell_1, x = -m, y = 0\rangle$ are simulation incomparable, when $n \neq m$. Hence there is no finite bisimulation.

We fix a safe GTA $\mathcal{A}$ for the rest of this section. We recall that our GTA are strongly non-Zeno, that is, every accepting run is non-Zeno. In order to focus on the main difficulties and avoid additional technicalities, we assume that the GTA $\mathcal{A}$ is without renamings. We start by noting that non-Zeno runs have a special form: future clocks which are not ultimately $-\infty$ should be released infinitely often. If not, there is a last point where a future clock is released to a finite value, and the entire suffix of the run should fall under this finite time, which contradicts non-Zenoness.

▶ **Lemma 5.** *Let $\rho := (q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \cdots$ be a non-Zeno run of the GTA $\mathcal{A}$. Then, for every future clock x of $\mathcal{A}$, and for every index $i \geq 0$, if $v_i(x) \neq -\infty$, there exists $j \geq i$ such that x is released in t_j.*

Overview of our solution. In classical timed automata, the liveness problem is solved by enumerating the zone graph, and using a *simulation equivalence* [23, 7, 16, 17] for termination: exploration from (q, Z) is stopped if there exists an already visited node (q, Z') such that $(q, Z) \preceq (q, Z')$ and $(q, Z') \preceq (q, Z)$ for some simulation relation $\preceq$. In this case a special edge is added between (q, Z) and (q, Z') to indicate a simulation equivalence. There is an (infinite) accepting run iff there is a cycle in the zone graph thus computed, containing an accepting state. The main point is that, from a cycle in the zone graph with simulation equivalences, we can conclude the existence of an infinite run over configurations.

At a high level the proof of this fact is as follows. Let us start by ignoring simulations for the moment: suppose $(q, Z) \xrightarrow{\sigma} (q, Z)$ for a sequence of transitions σ. By definition of successor computation in the zone graph, for every v in the zone Z (on the right), there exists a predecessor u in the zone Z (on the left). Repeatedly applying this argument gives a valuation $u \in Z$ from which σ can be iterated ℓ times, for any $\ell \geq 1$. When ℓ is greater than the number of Alur-Dill regions [3], we get a run $(q, u) \xrightarrow{\sigma^\ell} (q, u')$ such that u and u' are region equivalent. Since the region equivalence is a time-abstract bisimulation, this shows that we can once again do σ^ℓ from (q, u'), and so on. This leads to an infinite run from (q, u) where σ can be iterated infinitely often. Now, when simulations are involved, we need to consider sequences of the form $(q, Z) \xrightarrow{\sigma} (q, Z')$ where $(q, Z) \preceq (q, Z')$ and $(q, Z') \preceq (q, Z)$. An argument similar to the above can be adapted in this case too [28, 24, 22]. The critical underlying reason that makes such an argument possible is the presence of a finite time-abstract bisimulation, which in timed automata, is given by the region equivalence.

The same idea cannot be directly applied in the GTA setting, as there is no finite time-abstract bisimulation for GTAs, even with the safety assumption (Figure 2). However, [1] have defined a finite equivalence $v_1 \sim_M v_2$ and shown that the downward closures of the reachable zones w.r.t. a certain simulation called the G simulation [16, 17] are unions of $\sim_M$ equivalence classes. Therefore, applying an argument of the above style will give us a run $(q, u) \xrightarrow{\sigma^\ell} (q, u')$ such that $u \sim_M u'$. But we cannot conclude an infinite run immediately as $\sim_M$ is not a bisimulation.

To circumvent this problem, we will define an equivalence $\approx_M$ which is in spirit like the region equivalence in timed automata. As expected, $\approx_M$ will be a bisimulation. However, in accordance with the no finite timed-bisimulation result, $\approx_M$ will have an infinite index. We make a key observation: if we have a run $(q,u) \xrightarrow{\sigma} (q,u')$ such that $u \sim_M u'$ *and* if σ releases every future clock, then we can get a run $(q,u) \xrightarrow{\sigma} (q,u'')$ where $u \approx_M u''$ for a suitably modified valuation u''. Since $\approx_M$ is a bisimulation, this will then give an infinite run where σ can be iterated infinitely often. As we have seen from Lemma 5, if we are interested in non-Zeno runs, only such cycles where all future clocks are released (or remain $-\infty$) are relevant. Therefore, in order to decide liveness for safe GTAs, it suffices to construct the zone graph with the simulation equivalence edges and look for a reachable cycle that contains an accepting state such that for every future clock x, either x is released on the cycle, or valuation $-\infty$ is possible for clock x.

This section is organized as follows: we will first define the equivalence $\approx_M$ and show that it is a bisimulation; then we recall $\sim_M$, and prove the key observation mentioned above. One of the main challenges is in addressing diagonal constraints, which is exactly where the safety assumption is helpful.

A region-like equivalence for GTA. The definition of $\approx_M$ looks like the classical region equivalence extended from $[0,+\infty)$ to $\overline{\mathbb{R}}$: all clocks which are lesser than M (which automatically includes all future clocks) have the same integral values, and the ordering of fractional parts among these clocks is preserved. To account for diagonal constraints in guards, we explicitly add a condition to say that all allowed diagonal constraints are satisfied by equivalent valuations. This new equivalence does not have a finite index, but it turns out to be a time-abstract bisimulation, similar to the classical regions.

▶ **Definition 6.** *Let $v_1, v_2 \in \mathbb{V}$ be valuations. We say $v_1 \approx_M v_2$ if for all clocks x, y:*
1. *$v_1(x) \triangleleft c$ iff $v_2(x) \triangleleft c$ for all $\triangleleft \in \{<, \leq\}$ and $c \in \{-\infty, +\infty\}$ or $c \in \mathbb{Z}$ with $c \leq M$,*
2. *$v_1 \models x - y \triangleleft c$ iff $v_2 \models x - y \triangleleft c$ for all $\triangleleft \in \{<, \leq\}$ and $c \in \{-\infty, +\infty\}$ or $c \in \mathbb{Z}$ with $|c| \leq M$,*
3. *if $-\infty < v_1(x), v_1(y) \leq M$ then we have $\{v_1(x)\} \leq \{v_1(y)\}$ iff $\{v_2(x)\} \leq \{v_2(y)\}$.*

Notice that when $v_1 \approx_M v_2$, the first condition implies $v_1(x) = +\infty$ iff $v_2(x) = +\infty$, $v_1(x) = -\infty$ iff $v_2(x) = -\infty$, $-\infty < v_1(x) \leq M$ iff $-\infty < v_2(x) \leq M$, and in this case $\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$ and $\{v_1(x)\} = 0$ iff $\{v_2(x)\} = 0$.

▶ **Lemma 7.** *$\approx_M$ is a time-abstract bisimulation.*

The equivalence $\sim_M$, and moving from $\sim_M$ to $\approx_M$. The equivalence $\sim_M$ is defined on the space of all valuations. Our goal in this part is to start from $v_1 \sim_M v_2$ and generate a valuation v_2' by modifying some values of v_2, so that we get $v_1 \approx_M v_2'$. Let us first recall the definition of $\sim_M$, with n be the number of clocks in the GTA.

- First, we define $\sim_M$ on $\alpha, \beta \in \overline{\mathbb{R}}$ by $\alpha \sim_M \beta$ if $(\alpha \triangleleft c \iff \beta \triangleleft c)$ for all $\triangleleft \in \{<, \leq\}$ and $c \in \{-\infty, +\infty\}$ or $c \in \mathbb{Z}$ with $|c| \leq M$. In particular, $\alpha \sim_M \beta$ implies $\alpha = -\infty$ iff $\beta = -\infty$ and $\alpha = +\infty$ iff $\beta = +\infty$. Also, if $-M \leq \alpha \leq M$ then $\alpha \sim_M \beta$ implies $\lfloor \alpha \rfloor = \lfloor \beta \rfloor$ and $\{\alpha\} = 0$ iff $\{\beta\} = 0$.
- For valuations $v_1, v_2 \in \mathbb{V}$ we define $v_1 \sim_M v_2$ if (i) $v_1(x) \sim_{nM} v_2(x)$ for all $x \in X$, and (ii) $v_1(x) - v_1(y) \sim_{(n+1)M} v_2(x) - v_2(y)$ for all pairs of clocks $x, y \in X$.

Notice that $\approx_M$ and $\sim_M$ are incomparable, in the sense that neither of them is a refinement of the other. The equivalence $\approx_M$ constrains values up to M, whereas $\sim_M$ looks at values up to nM, i.e., between $-nM$ and nM. For instance, consider $v_1 := \langle x = M+2, y = 1 \rangle$ and

$v_2 := \langle x = M+3, y = 1 \rangle$ for some $M \geq 2$. We have $v_1 \approx_M v_2$, but $v_1 \not\sim_M v_2$. For the other way around, notice that $\sim_M$ has finite index, whereas $\approx_M$ does not. So, $v_1 \sim_M v_2$ does not imply $v_1 \approx_M v_2$. To see it more closely, $v_1 \approx_M v_2$ enforces the same integral values for all future clocks. For clocks less than $-(n+1)M$, there is no such constraint on the actual values in $\sim_M$.

As mentioned above, our objective is to obtain $\approx_M$ equivalent valuations starting from $\sim_M$ equivalent ones. Lemma 8 is a first step in this direction. It essentially shows that, when restricted to clocks within $-M$ and $+M$, $\sim_M$ entails $\approx_M$.

▶ **Lemma 8.** *Suppose $v_1 \sim_M v_2$. Let x, y be clocks such that $-M \leq v_1(x), v_1(y) \leq M$. Then, $\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$, $\{v_1(x)\} = 0$ iff $\{v_2(x)\} = 0$, and $\{v_1(x)\} \leq \{v_1(y)\}$ iff $\{v_2(x)\} \leq \{v_2(y)\}$.*

Lemma 8 considers clocks within $-M$ and $+M$. What about clocks above M? Directly from $v_1 \sim_M v_2$, we have $M < v_1(x)$ iff $M < v_2(x)$, and moreover diagonal constraints up to M are already preserved by $\sim_M$. Therefore, together with Lemma 8, $v_1 \sim_M v_2$ implies $v_1 \approx_M v_2$ when restricted to clocks greater than $-M$. We cannot say the same for clocks lesser than $-M$, in particular we may have $v_1(x) = -nM-1$ and $v_2(x) = -nM-2$. However, as shown in the lemma below, we can choose suitable values for clocks lesser than $-M$ to get a $\approx_M$-equivalent valuation from a $\sim_M$-equivalent one.

▶ **Lemma 9.** *Suppose $v_1 \sim_M v_2$, and let $L = \{x \mid -M \leq v_1(x)\}$. There is a valuation v_2' such that $v_2'\!\downarrow_L = v_2\!\downarrow_L$ and $v_1 \approx_M v_2'$.*

Finally, we show that if we have a run between $\sim_M$ equivalent valuations, we can extract a run between $\approx_M$ equivalent valuations, simply by changing the last released values of future clocks. Suppose there is a run ρ from configuration (q, v_1) to configuration (q, v_k) such that $v_1 \sim_M v_k$, and all future clocks are released in ρ. By Lemma 9, there is a v_k' satisfying $v_1 \approx_M v_k'$ that differs from v_k only in the clocks that are less than $-M$. In order to reach v_k' from v_1 using the same sequence of transitions as in ρ, it is enough to choose a suitable shifted value during the last release of the clocks that were modified. This gives a new run ρ'. The non-trivial part is to show that ρ' is indeed a run, that is: all guards are satisfied by the new values. We depict this situation in Figure 3. The modified clocks are those that are less than $-M$ in v_k. Clock x is one such. The black dot represents its value in v_k, and the blue dot is its value in v_k'. Its new value is still $< -M$. In the run ρ', clock x is released to a suitably shifted value at its last release point. Notice that from this last release point till k, clock x stays below $-M$ in both ρ and ρ'. Therefore, all non-diagonal constraints $x \triangleleft c$ that were originally satisfied in ρ continue to get satisfied in ρ'. Showing that all diagonal constraints are still satisfied is not as easy. Here, we make use of the safety assumption. Let us look at a diagonal constraint $x - y$, and a situation as in Figure 3 where the last release of y happens after the last release of x. For simplicity, let us assume there is no release of y in between these two points.

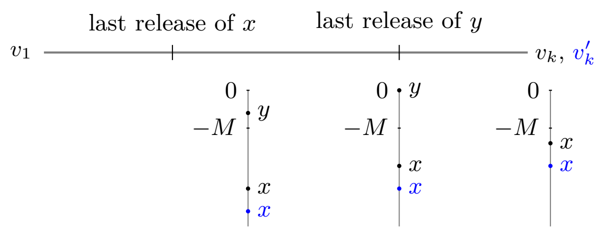

Figure 3 An illustration for the proof of Lemma 10.

We divide the run into three parts: the left part is the one before the last release of x, the middle part is the one between the two release points, and the right part is the rest of the run, to the right of the release of y. In the left part, the values of x and y are the same in both ρ and ρ', and so the diagonal constraints continue to get satisfied. In the right part, the value of $x - y$ equals $v'_k(x) - v'_k(y)$. Using $v'_k \approx_M v_1$ and $v_1 \sim_M v_k$, we can argue that v'_k and v_k satisfy the same diagonal constraints up to constant M. This takes care of the right part. The middle part is the trickiest. In this part, we know that x remains less than $-M$ in both ρ and ρ'. The value of y is the same in both ρ and ρ'. But what about the difference $x - y$? Can it be, say -1 in ρ and -2 in ρ'? Here is where we use the safety assumption to infer the value of $x - y$. Before y is released, its value should be 0. At that point, x is still less than $-M$ (in both the runs). Therefore $x - y < -M$ just before y is last released. As the differences do not change, we see that $x - y < -M$ in the middle part, for both runs. Hence the diagonal constraints continue to hold in ρ'. We formalize these observations in Lemma 10, where we exhaustively argue about all the different cases.

▶ **Lemma 10.** *Consider a safe GTA $\mathcal{A}$. Let $\rho : (q_1, v_1) \xrightarrow{\delta_1, t_1} (q_2, v_2) \xrightarrow{\delta_2, t_2} \cdots (q_k, v_k)$ be a run of $\mathcal{A}$ such that $v_1 \sim_M v_k$ and for every future clock x, either x is released in the transition sequence $t_1 \dots t_{k-1}$ or $v_1(x) = -\infty$. Let $L = \{x \mid -M \leq v_1(x)\}$. Let v'_k be a valuation such that $v'_k{\downarrow_L} = v_k{\downarrow_L}$ and $v_1 \approx_M v'_k$. Then, there exists a run of the form $\rho' : (q_1, v_1) = (q_1, v'_1) \xrightarrow{\delta_1, t_1} (q_2, v'_2) \xrightarrow{\delta_2, t_2} \cdots (q_k, v'_k)$ in $\mathcal{A}$, leading to (q_k, v'_k) from (q_1, v_1).*

We lift this argument to the level of zones, to obtain one of the main results of this paper.

▶ **Theorem 11.** *Let $(q, Z) = (q_1, Z_1) \xrightarrow{t_1} (q_2, Z_2) \xrightarrow{t_2} \cdots \xrightarrow{t_{k-1}} (q_k, Z_k) = (q, Z')$ be a run in the zone graph such that $(q, Z) \preceq (q, Z')$, $(q, Z') \preceq (q, Z)$ and for every future clock x, either x is released in the sequence $t_1 \dots t_{k-1}$, or there is a valuation $v_x \in Z'$ with $v_x(x) = -\infty$. Then, there is a valuation $v \in Z$ and an infinite run starting from (q, v) over the sequence of transitions $(t_1 \dots t_{k-1})^\omega$.*

Finally, combining Theorem 11 and Lemma 5, we get an algorithm for liveness: we construct the zone graph with simulation equivalence and check for a reachable cycle that contains an accepting state and where every future clock x which is not released during the cycle may take value $-\infty$ in some valuations of the zones in the cycle.

4 Translating MITL to GTA

We first introduce the preliminaries for Metric Interval Temporal Logic. Let Prop be a finite nonempty set of atomic propositions. The alphabet Σ that we consider is the set of subsets of Prop. The set of MITL formulae over the set of atomic propositions Prop is defined as

$$\varphi := p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathsf{X}_I\, \varphi \mid \varphi\, \mathsf{U}_I\, \varphi$$

where $p \in \mathsf{Prop}$, and I is either $[0, 0]$, or a non-singleton (open, or closed) interval whose end-points come from $\mathbb{N} \cup \{\infty\}$. In other words, if the end-points of the interval are a and b respectively, then either $a = b = 0$, or $a, b \in \mathbb{N} \cup \{\infty\}$ and $a < b$.

We will now define the *pointwise semantics* of MITL formulae inductively as follows. A timed word $w = (a_0, \tau_0)(a_1, \tau_1)(a_2, \tau_2) \cdots$ is said to satisfy the MITL formula φ at position $i \geq 0$, denoted as $(w, i) \models \varphi$ if (omitting the classical Boolean connectives)

- $(w, i) \models p$ if $p \in a_i$
- $(w, i) \models \mathsf{X}_I\, \varphi$ if $(w, i + 1) \models \varphi$ and $\tau_{i+1} - \tau_i \in I$.
- $(w, i) \models \varphi_1\, \mathsf{U}_I\, \varphi_2$ if there exists $j \geq i$ s.t. $(w, j) \models \varphi_2$, $(w, k) \models \varphi_1$ for all $i \leq k < j$, and $\tau_j - \tau_i \in I$.

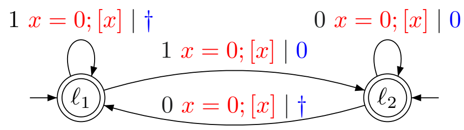

Figure 4 GTT for X_I using a future clock x. The output is 0 for transitions to location ℓ_2 and is the if-then-else $\dagger = (-x \in I)\,?\,1 : 0$ for transitions to location ℓ_1, where I is some interval.

Our goal is to construct a GTA with outputs for an MITL formula φ, which reads the timed word and outputs 1 at position i iff $(w, i) \models \varphi$. More precisely, there is a unique run of the GTA on w: $(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \cdots$, where the output of each transition t_i equals 1 iff $(w, i) \models \varphi$. We refer to GTA with outputs as Generalized Timed Tranducers (GTT) (discussed in detail in the full version [2]). At a high level, our construction can be viewed as structural induction on the *parse tree* of the MITL formula, where we build a GTT for atomic propositions, and then for each Boolean and temporal operator, and finally we compose these GTT bottom up to obtain the GTT for each subformula, which by structural induction finally gives us the GTT for the full formula. A detailed discussion of this compositional approach can be found in the full version [2]. We describe the transducers for X_I and U_I in this section.

Next operator. The transducer for $\mathsf{X}_I\, p$ is given in Figure 4. It is obtained by extending the untimed variant of the Next-transducer with a future clock x that predicts the time to the next event. The idea is the same as explained in Figure 1 of the Introduction. The prediction of the next event is verified, by having the guard $x = 0$ in every transition. Notice the use of the program syntax in this example: a transition first checks if $x = 0$ (satisfying a previous obligation), and then releases x to a non-deterministic value guessing the time to the next event, and then asks for a guard, either $-x \in I$ or $-x \notin I$.

Until operator. We start by describing the transducer for the untimed U modality $p\,\mathsf{U}\,q$ (in other words, $p\,\mathsf{U}_I\, q$ with $I = [0, \infty)$). This is shown in Figure 5. For simplicity, we have assumed $\mathsf{Prop} = \{p, q\}$ and the alphabet is represented as $(0, 0), (1, 0), (0, 1), (1, 1)$ corresponding to $\{\}$, $\{p\}$, $\{q\}$ and $\{p, q\}$. On the word w, if s_i is the state that reads a_i, then the following invariants hold:

- $s_i = q$ iff $q \in a_i$,
- $s_i = \neg q \wedge (p\,\mathsf{U}\,q)$ iff $q \notin a_i$ and $(w, i) \models p\,\mathsf{U}\,q$,
- $s_i = \neg(p\,\mathsf{U}\,q)$ iff $(w, i) \not\models p\,\mathsf{U}\,q$.

At the initial state the automaton makes a guess about position 0, and then subsequently on reading every a_i, it makes a guess about position $i + 1$ and moves to the corresponding state. The transitions implement this guessing protocol. For instance, transitions out of state q read letters with $q = 1$, and also output 1; transitions out of state $\neg(p\,\mathsf{U}\,q)$ have output 0. A noteworthy point is that state $q \wedge \neg(p\,\mathsf{U}\,q)$ is non-accepting, preventing the automaton to stay in that state forever. For every word, the transducer has a unique accepting run and the output at position i is 1 iff $(w, i) \models p\,\mathsf{U}\,q$.

Let us move on to the timed until U_I. Let us forget the specific interval I for the moment. We will come up with a generic construction, on which the outputs can be appropriately modified for specific intervals. To start the construction, we need the following notion.

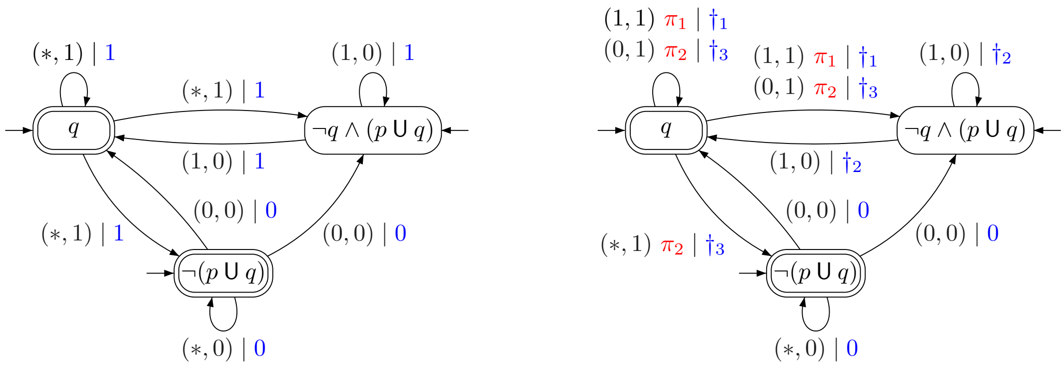

Figure 5 (Left) Transducer for the untimed LTL operator $p \, \mathsf{U} \, q$.
(Right) Transducer $\mathcal{A}$ tracking the earliest and last q witnesses for $p \, \mathsf{U} \, q$. Program π_1 is $x = 0; [x]$ and program π_2 is $x = 0; y = 0; [x, y]$. The outputs $\dagger_i$ depend on interval I in the timed until $p \, \mathsf{U}_I \, q$.

▶ **Definition 12.** *Let $w = (a_0, t_0)(a_1, t_1) \dots$ be a timed word and let $i \geq 0$. The* earliest *q-witness at position i is the least position $j > i$ such that $q \in a_j$, if it exists. We denote this position j giving the earliest q-witness at i as i_f. The* last *q-witness is the* least *position $j > i$ that satisfies*

$$\alpha = q \wedge \neg(p \wedge \mathsf{X}(p \, \mathsf{U} \, q)) \equiv (q \wedge \neg p) \vee (q \wedge \mathsf{X} \, \neg(p \, \mathsf{U} \, q))$$

We denote this position j giving the last q-witness at i as i_ℓ.

The earliest and last q-witnesses provide a convenient mechanism to check $p \, \mathsf{U}_I \, q$ which, in many cases, can be deduced by knowing the time to the earliest and last witnesses. Figure 6 illustrates the interpretation of x and y.

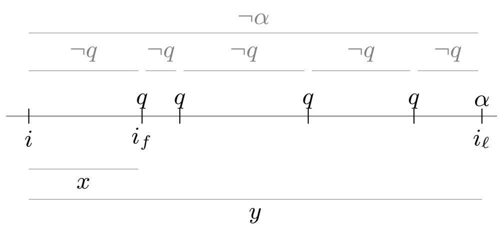

Figure 6 Division of q events, and interpretation of x, y.

Our next task is to extend the U transducer of Figure 5 to include two future clocks x and y that predict at each i, the time to i_f and i_ℓ, respectively. Figure 5 describes the transducer $\mathcal{A}$. For clock x to maintain time to i_f at each position i, we can do the following: at every transition that reads q, the transducer checks for $x = 0$ as guard and releases x (with the time to the next q). If there is no such q, then x needs to be released to $-\infty$ in order to continue the run, as our timed words are non-Zeno. Transitions satisfying $\neg q$ do not check for a guard on x or release x. Therefore, in any run, the value of x determines the time to the next q event.

In Figure 6, the last witness (property α) can be identified by transitions of the form $(0, 1)$ (signifying $q \wedge \neg p$) and transitions $(*, 1)$ going to state $\neg(p \, \mathsf{U} \, q)$ (for $q \wedge \mathsf{X} \, \neg(p \, \mathsf{U} \, q)$). Similar to the previous case of the earliest witness, every time we see such a transition we check for $y = 0$ as a guard and release y. No other transition checks or updates y. Notice that only the transitions with q have been changed. All transitions $(0,1)$ check and release both clocks (program π_2). Transitions $(1,1)$ that do not go to $\neg(p \mathsf{U} q)$ check and release only x (program π_1), whereas the $(*,1)$ transition that goes to $\neg(p \mathsf{U} q)$ does π_2.

▶ **Lemma 13.** *For every timed word $w = (a_0,\tau_0)(a_1,\tau_1)\cdots$, there is a unique run of $\mathcal{A}$ of the form: $(s_0,v_0) \xrightarrow{\tau_0,\theta_0} (s_1,v_1) \xrightarrow{\tau_1-\tau_0,\theta_1} \cdots$ such that for every position $i \geq 0$: (1) s_i is state q of $\mathcal{A}$ iff $w,i \models q$, (2) s_i is state $\neg q \wedge (p \mathsf{U} q)$ iff $w,i \models \neg q \wedge (p \mathsf{U} q)$, (3) s_i is state $\neg(p \mathsf{U} q)$ iff $w,i \models \neg(p \mathsf{U} q)$, (4) $v_i(x) = \tau_i - \tau_{i_f}$ and $v_i(y) = \tau_i - \tau_{i_\ell}$.*

Using $\mathcal{A}$ we can already answer $p \mathsf{U}_I q$ for one-sided intervals: $[0,c], [0,c), [b,+\infty), (b,+\infty)$, for natural numbers b, c.

- if $0 \in I$: $\dagger_1 = \dagger_3 = 1$ (current position is a witness), and $\dagger_2 = (-x \in I \vee -y \in I)\,?\,1 : 0$,
- if $0 \notin I$: $\dagger_3 = 0$, and $\dagger_1 = \dagger_2 = (-x \in I \vee -y \in I)\,?\,1 : 0$.

This is because in one-sided intervals, if at all there is a witness, the earliest or the last is one of them.

Until with a non-singular interval. We will now deal with the case of intervals $I = [b,c]$ with $0 < b < c < \infty$. Firstly, using x and y, some easy cases of $p \mathsf{U}_I q$ can be deduced. Output remains 0 for transitions starting from $\neg(p \mathsf{U} q)$. For other transitions, here are some extra checks:

- if $-x \in I$ or $-y \in I$, output 1 (one of the earliest or last witness is also a witness for $p \mathsf{U}_I q$),
- else, if $-y < b$ or $c < -x$, output 0 (the time to the last witness is too small or the time to the earliest witness is too large, so there is no witness within I).

If neither of the above cases hold, then we need guess a potential witness within $[b,c]$ and verify it. This requires substantial book-keeping which we will now explain. Assume we are given a timed word $w = (a_0,t_0)(a_1,t_1)\cdots$. Let us a call $j \geq 0$ a *difficult point* if:

$$w,j \models p \mathsf{U} q \text{ and } t_{j_f} < t_j + b \text{ and } t_j + c < t_{j_\ell}$$

This leaves the possibility for a q-witness within $[b,c]$. So, for difficult points, we need to make a prediction whether we have a q-witness within $[t_j+b, t_j+c]$: guess a time to a witness within $[t_j+b, t_j+c]$ and check it. We cannot keep making such predictions for every difficult point as we have only finitely many clocks. Therefore, we will guess some special witnesses. First we state a useful property.

▶ **Lemma 14.** *Let j be a difficult point. Then, for all k such that $j \leq k \leq j_\ell$, we have $w,k \models p \mathsf{U} q$.*

Therefore, automaton $\mathcal{A}$ stays in the top two states, while reading j upto j_ℓ.

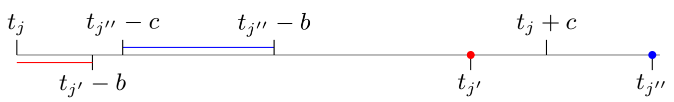

Figure 7 Illustration of a point j. The point j' is the last q-witness before $t_j + c$, and j'' is the first q-witness after $t_j + c$.

We will now come back to the idea of choosing special witnesses. This is illustrated in Figure 7. For a point j, we let $j' \geq j$ be the greatest position containing q such that $t_{j'} \leq t_j + c$. Let $j'' > j$ be the least position containing q such that $t_j + c < t_{j''}$. So, no

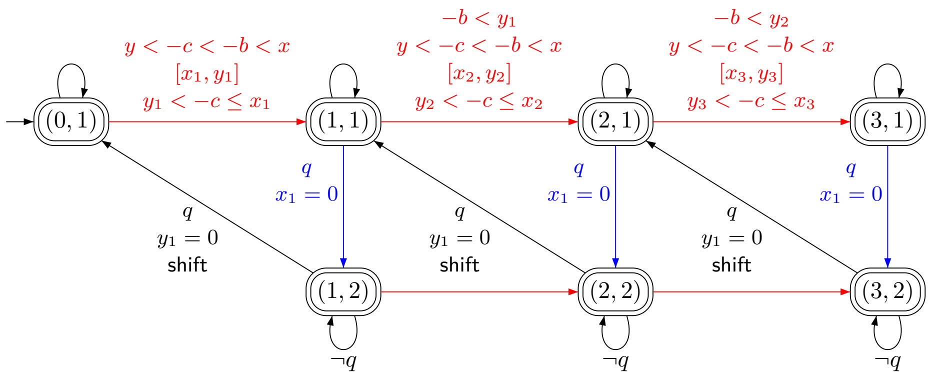

Figure 8 The automaton $\mathcal{B}$ for predicting q-witnesses which are not given by the earliest and latest. For clarity, not every transition is indicated. All clocks are future clocks.

position $j' < k < j''$ contains q. While reading a difficult point j, let us make use of fresh clocks x_1 and y_1 to predict these two witnesses:

$$x_1 = t_j - t_{j'} \qquad y_1 = t_j - t_{j''}$$

For the next important observation, we will once again take the help of Figure 7. Notice that for all points i with $t_i \in [t_j, t_{j'} - b]$, the point j' is also a witness for $w, i \models p \, \mathsf{U}_{[b,c]} \, q$. Similarly, j'' is a witness for all i such that $t_i \in [t_{j''} - c, t_{j''} - c]$. Therefore for all i such that $t_i \in [t_j, t_{j''} - b]$, we have a way to determine the output: it is 1 iff while reading a_i we have $-x_1 \in I$ or $-y_1 \in I$ (recall we have predicted x_1 and y_1 while reading a_j as explained above). So, we do not have to make new guesses at the difficult points in $[t_j, t_{j''} - b]$. After $t_{j''} - b$ (which can be identified with the constraint $-b < y_1$), we need to make new such guesses, using fresh clocks, say x_2, y_2. We will call the difficult points where we start new guesses as *special difficult points*. Notice that the distance between two special difficult points is at least $c - b$ (which is ≥ 1, as we consider non-singular intervals with bounds in $\mathbb{N}$). In the figure, if j is a special point, a new special point will be opened later than $t_{j''} - b$.

This gives a bound on the number of special points that can be open between j and j''. Suppose $j < \ell_1 < \ell_2 < \cdots < \ell_i < j''$ be the sequence of special points between j and j''. Since ℓ_1 is opened when time to j'' is atmost b, we get the inequality: $t_{\ell_i} - t_{\ell_1} < b$. Since consecutive special points are at least $c - b$ apart, we have $(i-1)(c-b) < b$. This entails $i < 1 + \lceil \frac{b}{c-b} \rceil$. By the time we reach j'', we need to have opened at most $k = 1 + \lceil \frac{b}{c-b} \rceil$ special points, and hence we can work with the extra clocks $x_1, y_1, x_2, y_2, \ldots, x_k, y_k$.

All these ideas culminate in a book-keeping automaton $\mathcal{B}$ to handle difficult points. Its set of states is $\{0, 1, \ldots, N\} \times \{1, 2\}$ where $N = 1 + \lceil \frac{b}{c-b} \rceil$ (state $(0,2)$ is not reachable). All states are accepting. This is shown in Figure 8 for $N = 3$. The automaton $\mathcal{B}$ synchronizes with $\mathcal{A}$ (via a usual cross-product synchronized on transitions). All transitions of $\mathcal{B}$ other than the self loop on state $(0,1)$ satisfy p. Transitions which satisfy q, and $\neg q$ are specifically marked in the figure.

The automaton $\mathcal{B}$ starts in the initial state $(0,1)$. It moves to $(1,1)$ on the first difficult point j (which will become special) and comes back to $(0,1)$ when there are no active special difficult points waiting for witnesses (a special difficult point j is active at positions $j \leq i \leq j''$). States $(i,1)$ in the top indicate that there are i active special difficult points currently. A state $(i,2)$ indicates that the j' witness for the oldest active point has been seen, and we are waiting for its j'' witness (the space between $t_{j'}$ and $t_{j''}$ in Figure 7).

The red transitions $(i,*) \to (i+1,*)$ open new special difficult points, and contain the program as illustrated in the figure. At $(i,1)$, suppose $\ell_1 < \ell_2 < \cdots < \ell_i$ are the active special difficult points where we have predicted $x_1, y_1, \dots, x_i, y_i$ respectively. We have the invariant:

$$y_i < x_i \le y_{i-1} < x_{i-1} \le \cdots \le y_2 < x_2 \le y_1 < x_1$$

Notice that we may have $x_i = y_{i-1}$: the "first" witness of the i^{th} special point (ℓ'_i) could coincide with the "second" witness of the $(i-1)^{th}$ point (ℓ'_{i-1}). This leads to certain subtleties, which we will come to later.

The blue transitions read the witness for the oldest active special point (that is, we have reached ℓ'_1). Observe that $x_1 = 0$ does not immediately identify ℓ'_1, since there could be a sequence of positions at the same time, and ℓ'_1 is the last of them. Therefore, we make a non-deterministic choice whether to take the blue transition (implying that ℓ'_1 has been found), or we remain in the same state. The blue transitions read a q, check $x_1 = 0$, and then releases x_1 to $-\infty$ (not shown in Figure 7). The black (diagonal) transitions witness ℓ''_1. When this happens, x_1, y_1 are no longer useful, and therefore all the higher clocks are shifted using the permutation shift which maps $x_2, y_2, \dots x_k, y_k, x_1, y_1$ to $x_1, y_1, \dots, x_k, y_k$ and keeps the other clocks unchanged.

There are some subtleties which arise when special points coincide with witness points, or when the second witness of a special point coincides with the first witness of the consecutive special point.

Subtleties. The first subtlety arises when we have $\ell''_j = \ell'_{j+1}$ for consecutive special points. This will imply $y_j = x_{j+1}$. The reverse direction is not true, as there could be a sequence of positions with the same time, but let us assume we have dealt with it by the non-deterministic choice. When we actually witness these points, the clock values would have shifted to lower indices. This situation will be manifested as $y_1 = x_2 = 0$. Suppose we are in $(i,2)$ and see a point ℓ''_j ($y_1 = 0$). The diagonal transition takes the automaton to $(i-1,1)$ and shifts x_2 to x_1. Now, $x_1 = 0$ (as $\ell'_{j+1} = \ell''_j$). Therefore, we will have to combine the black-diagonal-left with the downward-blue to get the combined effect. This leads to these two divisions:

$$(i,2) \xrightarrow{y_1=0} (i-1,1) \qquad\qquad (i,2) \xrightarrow{y_1=0 \wedge x_2=0} (i-1,2)$$

The second subtlety is that one of either ℓ'_j or ℓ''_j witnesses be a new special point (notice that the red transitions are independent of the blue and black transitions). In such cases, we can combine the two effects in any order: first discharge x_1 or y_1 verification, and then open a new special point or vice-versa. This leads to some additional divisions of the form:

$$(i,1) \xrightarrow{x_1=0 \wedge -b<y_i} (i+1,2) \qquad\qquad (i,2) \xrightarrow{y_1=0 \wedge -b<y_i} (i,1)$$

In the first transition, we have combined a blue and a red (in any order); whereas in the second, we have combined a red and a black-diagonal, in any order.

The third subtlety is that the first and second subtleties may occur together! A point could be ℓ''_j, ℓ'_{j+1} and also a new special point. We illustrate this on a specific state $(i,2)$. We provide only the "guards". The full program is obtained by suitably combining the effects of the individual transitions:

$$(i,2) \xrightarrow{y_1=0 \wedge y_i \le -b} (i-1,1) \qquad\qquad (i,2) \xrightarrow{y_1=0 \wedge -b<y_i} (i,1)$$

$$(i,2) \xrightarrow{y_1=0 \wedge x_2=0 \wedge y_i \le -b} (i-1,2) \qquad\qquad (i,2) \xrightarrow{y_1=0 \wedge x_2=0 \wedge -b<y_i} (i,2)$$

This concludes the description of the automaton $\mathcal{B}$. The product $\mathcal{A} \times \mathcal{B}$ gives the required transducer for $p \,\mathsf{U}_I\, q$. The full construction of $\mathcal{B}$, taking into account all these subtleties, is described as Algorithm 1. In comments, we use the terminology introduced before and we also refer to the color of transitions in Figure 8. Parsing the pseudo-code from a current state (k, m) results in a sequence of guards and releases, an output of a Boolean value (output value), and the next state (goto (k', m')). The most difficult case is for states $(k, 2)$ with $k \geq 2$, where we could generate transitions to states $(k-1, 1), (k-1, 2), (k, 1), (k, 2), (k+1, 2)$.

Complexity and comparison with the MightyL approach. The final automaton $\mathcal{A} \times \mathcal{B}$ has at most $6k$ states, where $k = 1 + \lceil \frac{b}{c-b} \rceil$ as defined above: automaton $\mathcal{A}$ has 3 states, and automaton $\mathcal{B}$ has $2k - 1$ states (see Figure 8). In terms of clocks, $\mathcal{A}$ has 2 future clocks x, y, and $\mathcal{B}$ has $2k$ future clocks $x_1, y_1, \ldots, x_k, y_k$. We have used a permutation operation shift. As we mention in Remark 2, renamings can be eliminated by maintaining in the current state the composition of permutations applied since the initial state. Since each permutation does a cyclic shift, in any composition, the clocks $x_1, y_1, \ldots, x_k, y_k$ are renamed to some $x_i, y_i, \ldots, x_k, y_k, x_1, y_1, \ldots, x_{i-1}, y_{i-1}$. Therefore, there are at most k renamings. Maintaining them in states gives rise to atmost $\mathcal{O}(k^2)$ states.

In contrast, the state-of-the-art approach [11] starts with a 1-clock alternating timed automata for U_I. After reading a timed word, the 1-ATA reaches a configuration containing several state-valuation pairs (q, v). A finite abstraction of this set of configurations, called the interval semantics, has been proposed [9, 10, 11]. This abstraction is maintained in the states. Overall, the number of locations for $p \,\mathsf{U}_I\, q$ is exponential in k, and the number of clocks is $2k + 2$.

Due to the presence of future clocks, we are able to make predictions, as in Figure 7 and the GTA syntax enables concisely checking these predictions in the transitions. Therefore, we are able to give a direct construction to the final automaton, instead of going via an alternating automaton and then abstracting it.

5 Conclusion

In this paper, we have answered two problems: (1) liveness of GTA and (2) MITL model checking using GTA. The solution to the first problem required to bypass the technical difficulty of having no finite time-abstract bisimulation for GTAs. The presence of diagonal constraints adds additional challenges. For MITL model checking using GTA, we have described the GTA for the X_I and U_I modalities. Indeed, the presence of future clocks allows to make predictions better and we see an exponential gain over the state-of-the-art, in the number of states of the final automaton produced. Moreover, our construction is direct, without having to go via alternation.

The next logical step would be to implement these ideas and see how they perform in practice, and compare them with existing well-engineered tools (e.g., [11]). This will require a considerable implementation effort, needing several optimizations and incorporating of many practical considerations before it can become scalable. This provides tremendous scope for future work on these lines.

Algorithm 1 Automaton $\mathcal{B}$ (synchronized with $\mathcal{A}$).

```
State $(0,1)$:                                                    ▷ initial state of $\mathcal{B}$
    if $\mathcal{A}$ at state $\neg(p \mathsf{U} q)$ then output 0; goto (0,1) end if
    if $-x \in I$ or $-y \in I$ then output 1; goto (0,1) end if
    if $x < -c$ or $-b < y$ then output 0; goto (0,1) end if
    Release $[x_1, y_1]$                                          ▷ Special difficult point
    Check $y_1 < -c \leq x_1$
    output $(x_1 \leq -b)$                                        ▷ Boolean value
    goto (1,1)                                                    ▷ red transition
State $(k,1)$ with $k > 0$:                                       ▷ waiting for the event predicted by $x_1$
    $k' \leftarrow k$
    if $y_k \leq -b$ then
        output $(x_k \in I) \vee (y_k \in I)$                     ▷ Boolean value
    else
        if $-c \leq y$ then                                       ▷ not a difficult point
            output $(y \leq -b)$                                  ▷ Boolean value $(y \in I)$
        else                                                      ▷ new special difficult point, red transition,
                                                                  ▷ possibly combined with a blue transition below
            $k' \leftarrow k+1$;
            Release $[x_k, y_k]$; Check $y_k < -c \leq x_k$
            output $(x_k \leq -b)$                                ▷ Boolean value
        end if
    end if
    choose non-deterministically
        when True do goto $(k',1)$                                ▷ not the event predicted by $x_1$
        when $q \wedge (x_1 = 0)$ do Release $[x_1]$; $x_1 = -\infty$; goto $(k',2)$
                                                                  ▷ blue transition
    end choose
State $(k,2)$ with $k > 0$:                                       ▷ waiting for the event predicted by $y_1$
    $k' \leftarrow k$
    if $y_k \leq -b$ then
        output $(x_k \in I) \vee (y_k \in I)$                     ▷ Boolean value
    else
        if $-c \leq y$ then                                       ▷ not a difficult point
            output $(y \leq -b)$                                  ▷ Boolean value $(y \in I)$
        else                                                      ▷ new special difficult point, red transition,
                                                                  ▷ possibly combined with a black transition below
            $k' \leftarrow k+1$;
            Release $[x_k, y_k]$; Check $y_k < -c \leq x_k$
            output $(x_k \leq -b)$                                ▷ Boolean value
        end if
    end if
    if $\neg q$ then                                              ▷ not the event predicted by $y_1$
        goto $(k',2)$
    else                                                          ▷ event predicted by $y_1$, black transition
                                                                  ▷ possibly combined with a blue transition below
        Check $y_1 = 0$; Release $[y_1]$; $y_1 = -\infty$
        if $k' = 1$ then
            goto $(0,1)$
        else
            Shift $x_2, y_2, \ldots, x_k, y_k, x_1, y_1$ to $x_1, y_1, \ldots, x_k, y_k$
        end if
        choose non-deterministically
            when True do goto $(k'-1,1)$
                                                                  ▷ not the event predicted by the new $x_1$
            when $(x_1 = 0)$ do $Release[x_1]$; $x_1 = -\infty$; goto $(k'-1,2)$
                                                                  ▷ blue transition
        end choose
    end if
```

References

1 S. Akshay, Paul Gastin, R. Govind, Aniruddha R. Joshi, and B. Srivathsan. A unified model for real-time systems: Symbolic techniques and implementation. In *CAV (1)*, volume 13964 of *Lecture Notes in Computer Science*, pages 266–288. Springer, 2023.

2 S. Akshay, Paul Gastin, R. Govind, and B. Srivathsan. MITL model checking via generalized timed automata and a new liveness algorithm, 2024. `arXiv:2407.08452`.

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

4 Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.

5 Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999.

6 Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. In *LICS*, pages 390–401. IEEE Computer Society, 1990.

7 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.

8 Laura Bozzelli, Angelo Montanari, and Adriano Peron. Complexity issues for timeline-based planning over dense time under future and minimal semantics. *Theor. Comput. Sci.*, 901:87–113, 2022.

9 Thomas Brihaye, Morgane Estiévenart, and Gilles Geeraerts. On MITL and alternating timed automata. In *FORMATS*, volume 8053 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2013.

10 Thomas Brihaye, Morgane Estiévenart, and Gilles Geeraerts. On MITL and alternating timed automata over infinite words. In *FORMATS*, volume 8711 of *Lecture Notes in Computer Science*, pages 69–84. Springer, 2014.

11 Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. MightyL: A compositional translation from MITL to timed automata. In *CAV (1)*, volume 10426 of *Lecture Notes in Computer Science*, pages 421–440. Springer, 2017.

12 Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model verifier. In *CAV*, volume 1633 of *Lecture Notes in Computer Science*, pages 495–499. Springer, 1999.

13 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.

14 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - A framework for LTL and ω-automata manipulation. In *ATVA*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129, 2016.

15 Thomas Ferrère, Oded Maler, Dejan Nickovic, and Amir Pnueli. From real-time logic to timed automata. *J. ACM*, 66(3):19:1–19:31, 2019.

16 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in timed automata with diagonal constraints. In *CONCUR*, volume 118 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

17 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 41–59. Springer, 2019.

18 Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.

19 Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. On regions and zones for event-clock automata. *Formal Methods Syst. Des.*, 45(3):330–380, 2014.

20 Rob Gerth, Doron A. Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV*, volume 38 of *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, 1995.

21 F. Herbreteau and G. Point. TChecker. `https://github.com/fredher/tchecker`, v0.2 - April 2019.

22 Frédéric Herbreteau, B. Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz. Why liveness for timed automata is hard, and what we can do about it. *ACM Trans. Comput. Log.*, 21(3):17:1–17:28, 2020. `doi:10.1145/3372310`.

23 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. In *LICS*, pages 375–384. IEEE Computer Society, 2012.

24 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Efficient emptiness check for timed büchi automata. *Formal Methods Syst. Des.*, 40(2):122–146, 2012.

25 Gerard J. Holzmann. The model checker SPIN. *IEEE Trans. Software Eng.*, 23(5):279–295, 1997.

26 Henrik Ejersbo Jensen, Kim G. Larsen, and Arne Skou. Modelling and analysis of a collision avoidance protocol using SPIN and UPPAAL. In *The Spin Verification System*, volume 32 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 33–49. DIMACS/AMS, 1996.

27 Leslie Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

28 Guangyuan Li. Checking timed Büchi automata emptiness using LU-abstractions. In Joël Ouaknine and Frits W. Vaandrager, editors, *Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings*, volume 5813 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2009. `doi:10.1007/978-3-642-04368-0_18`.

29 Oded Maler, Dejan Nickovic, and Amir Pnueli. From MITL to timed automata. In *FORMATS*, volume 4202 of *Lecture Notes in Computer Science*, pages 274–289. Springer, 2006.

30 Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2018.

31 Joël Ouaknine and James Worrell. On metric temporal logic and faulty turing machines. In *FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.

32 Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

33 Amir Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In *Current Trends in Concurrency*, volume 224 of *Lecture Notes in Computer Science*, pages 510–584. Springer, 1986.

34 Amir Pnueli and Eyal Harel. Applications of temporal logic to the specification of real-time systems. In *FTRTFT*, volume 331 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 1988.

35 Stavros Tripakis, Sergio Yovine, and Ahmed Bouajjani. Checking timed Büchi automata emptiness efficiently. *Formal Methods Syst. Des.*, 26(3):267–292, 2005.

36 Moshe Y Vardi. An automata-theoretic approach to linear temporal logic. *Lecture Notes in Computer Science*, 1043:238–266, 1996.

37 Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *FTRTFT*, volume 863 of *Lecture Notes in Computer Science*, pages 694–715. Springer, 1994.