Logical Methods in Computer Science
Vol. 11(3:20)2015, pp. 1–24
www.lmcs-online.org

Submitted Jul. 1, 2014
Published Sep. 22, 2015

VERIFICATION FOR TIMED AUTOMATA EXTENDED WITH UNBOUNDED DISCRETE DATA STRUCTURES

KARIN QUAAS

Universität Leipzig, Germany
e-mail address: quaas@informatik.uni-leipzig.de

Abstract. We study decidability of verification problems for timed automata extended with unbounded discrete data structures. More detailed, we extend timed automata with a pushdown stack. In this way, we obtain a strong model that may for instance be used to model real-time programs with procedure calls. It is long known that the reachability problem for this model is decidable. The goal of this paper is to identify subclasses of timed pushdown automata for which the language inclusion problem and related problems are decidable.

1. Introduction

Timed automata were introduced by Alur and Dill [5], and have since then become a popular standard formalism to model real-time systems. An undeniable reason for the success of timed automata is the PSPACE decidability of the *language emptiness problem* [5]. A major drawback of timed automata is the undecidability [5] of the *language inclusion problem*: given two timed automata $\mathcal{A}$ and $\mathcal{B}$, does $L(\mathcal{A}) \subseteq L(\mathcal{B})$ hold? The undecidability of this problem prohibits the usage of automated verification algorithms for analysing timed automata, where $\mathcal{B}$ can be seen as the specification that is supposed to be satisfied by the system modelled by $\mathcal{A}$. However, if $\mathcal{B}$ is restricted to have at most one clock, then the language inclusion problem over finite timed words is decidable (albeit with non-primitive recursive complexity) [34]. As other important milestones in the success story of timed automata we would like to mention the decidability of bisimulation [41], the decidability of the *model checking problem* for timed automata and a timed extension of CTL [24], and, more recently, the decidability of the model checking problem for timed automata and Metric Temporal Logic (MTL, for short) over finite timed words [35].

Timed automata can express many interesting time-related properties, and even with the restriction to a single clock, they allow one to model a large class of systems, including,

2012 ACM CCS: [**Theory of computation**]: Models of Computation; Formal languages and automata theory—Automata over infinite objects.

Key words and phrases: Timed automata, real-time systems, counter systems, pushdown automata, Metric Temporal Logic.

[*] An extended abstract was published in *CONCUR 2014*.

The author is supported by DFG, project QU 316/1-1.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-11(3:20)2015

for example, the internet protocol TCP [33]. If we want to reason about real-time programs with procedure calls, or about the number of events occurring in computations of real-time systems, we have to extend the model of timed automata with unbounded discrete data structures. In 1994, Bouajjani et al. [8] extended timed automata with discrete counters and a pushdown stack, and proved that the satisfiability of reachability properties for several subclasses of this model is decidable. Nine years later, it was shown that the binary reachability relation for *timed pushdown systems* is decidable [16]. Decidability of the reachability problem was also proved for several classes of *timed counter systems* [9], mainly by simple extensions of the classical region-graph construction [5]. The language inclusion problem, however, is to the best of our knowledge only considered in [20] for the class of timed pushdown systems. In [20] it is stated that the language inclusion problem is decidable if $\mathcal{A}$ is a timed pushdown automaton, and $\mathcal{B}$ is a one-clock timed automaton. The proof is based on an extension of the proof for the decidability of the language inclusion problem for the case that $\mathcal{A}$ is a timed automaton without pushdown stack [34]. Unfortunately, and as is well known, the proof in [20] is not correct.

In this paper, we prove that different to what is claimed in [20], the language inclusion problem for the case that $\mathcal{A}$ is a pushdown timed automaton and $\mathcal{B}$ is a one-clock timed automaton is undecidable. This is even the case if $\mathcal{A}$ is a deterministic instance of a very restricted subclass of timed pushdown automata called *timed visibly one-counter nets*. On the other hand, we prove that the language inclusion problem is decidable if $\mathcal{A}$ is a timed automaton and $\mathcal{B}$ is a timed automaton extended with a finite set of counters that can be incremented and decremented, and which we call *timed counter nets*. As a special case, we obtain the decidability of the *universality problem* for timed counter nets: given a timed automaton $\mathcal{B}$ with input alphabet Σ, does $L(\mathcal{B})$ accept the set of all timed words over Σ? Finally, we give the precise decidability border for the universality problem by proving that the universality problem is undecidable for the class of *timed visibly one-counter automata*. We remark that all results apply to extensions of timed automata over *finite* timed words.

2. Extensions of Timed Automata with Discrete Data Structure

We use $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ to denote the integers, the non-negative integers and the non-negative reals, respectively.

We use Σ to denote a finite alphabet. A *timed word* over Σ is a non-empty finite sequence $(a_1, t_1) \dots (a_k, t_n) \in (\Sigma \times \mathbb{R}_{\geq 0})^+$ such that the sequence $t_1, \dots, t_n$ of timestamps is non-decreasing. We say that a timed word is *strictly monotonic* if $t_{i-1} < t_i$ for every $i \in \{2, \dots, n\}$. We use $T\Sigma^+$ to denote the set of finite timed words over Σ. A set $L \subseteq T\Sigma^+$ is called a *timed language*.

Let $\mathcal{X}$ be a finite set of *clock variables* ranging over $\mathbb{R}_{\geq 0}$. We define *clock constraints* ϕ over $\mathcal{X}$ to be conjunctions of formulas of the form $x \sim c$, where $x \in \mathcal{X}$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$. We may use $\texttt{true}$ as abbreviation for $x \leq c \vee x > c$. We use $\Phi(\mathcal{X})$ to denote the set of all clock constraints over $\mathcal{X}$. For the case that $\mathcal{X}$ is the empty set, we set $\Phi(\mathcal{X}) = \{\texttt{true}\}$. A *clock valuation* is a mapping from $\mathcal{X}$ to $\mathbb{R}_{\geq 0}$. A clock valuation ν satisfies a clock constraint ϕ, written $\nu \models \phi$, if ϕ evaluates to true according to the values given by ν. For $\delta \in \mathbb{R}_{\geq 0}$ and $\lambda \subseteq \mathcal{X}$, we define $\nu + \delta$ to be $(\nu + \delta)(x) = \nu(x) + \delta$ for each $x \in \mathcal{X}$, and we define $\nu[\lambda := 0]$ by $(\nu[\lambda := 0])(x) = 0$ if $x \in \lambda$, and $(\nu[\lambda := 0])(x) = \nu(x)$ otherwise.

Let Γ be a finite stack alphabet. We use Γ^* to denote the set of finite words over Γ, including the empty word denoted by ε. We define a finite set $\mathsf{Op}(\Gamma)$ of *stack operations* by $\mathsf{Op}(\Gamma) := \{pop(a), push(a) \mid a \in \Gamma\} \cup \{noop, empty?\}$.

A *timed pushdown automaton* is a tuple $\mathcal{A} = (\Sigma, \Gamma, \mathcal{L}, \mathcal{L}_0, \mathcal{L}_f, \mathcal{X}, E)$, where

- $\mathcal{L}$ is a finite set of *locations*,
- $\mathcal{L}_0 \subseteq \mathcal{L}$ is the set of initial locations,
- $\mathcal{L}_f \subseteq \mathcal{L}$ is the set of accepting locations,
- $E \subseteq \mathcal{L} \times \Sigma \times \Phi(\mathcal{X}) \times \mathsf{Op}(\Gamma) \times 2^{\mathcal{X}} \times \mathcal{L}$ is a finite set of edges.

A *state* of $\mathcal{A}$ is a triple (l, ν, u), where $l \in \mathcal{L}$ is the current location, the clock valuation ν represents the current values of the clocks, and $u \in \Gamma^*$ represents the current stack content, where the top-most symbol of the stack is the left-most symbol in the word u, and the empty word ε represents the empty stack. We use $\mathcal{G}^{\mathcal{A}}$ to denote the set of all states of $\mathcal{A}$. A timed pushdown automaton $\mathcal{A}$ induces a transition relation $\Rightarrow_{\mathcal{A}}$ on $\mathcal{G}^{\mathcal{A}} \times \mathbb{R}_{\geq 0} \times \Sigma \times \mathcal{G}^{\mathcal{A}}$ as follows: $\langle (l, \nu, u), \delta, a, (l', \nu', u') \rangle \in \Rightarrow_{\mathcal{A}}$, if, and only if, there exists some edge $(l, a, \phi, \mathsf{op}, \lambda, l') \in E$ such that $(\nu + \delta) \models \phi$, $\nu' = (\nu + \delta)[\lambda := 0]$, and (i) if $\mathsf{op} = pop(\gamma)$ for some $\gamma \in \Gamma$, then $u = \gamma \cdot u'$; (ii) if $\mathsf{op} = push(\gamma)$ for some $\gamma \in \Gamma$, then $u' = \gamma \cdot u$; (iii) if $\mathsf{op} = empty?$, then $u = u' = \varepsilon$; (iv) if $\mathsf{op} = noop$, then $u' = u$. A *run* of $\mathcal{A}$ is a finite sequence $\prod_{1 \leq i \leq n} \langle (l_{i-1}, \nu_{i-1}, u_{i-1}), \delta_i, a_i, (l_i, \nu_i, u_i) \rangle$ such that $\langle (l_{i-1}, \nu_{i-1}, u_{i-1}), \delta_i, a_i, (l_i, \nu_i, u_i) \rangle \in \Rightarrow_{\mathcal{A}}$ for every $i \in \{1, \ldots, n\}$. A run is called *successful* if $l_0 \in \mathcal{L}_0$, $\nu_0(x) = 0$ for every $x \in \mathcal{X}$, $u_0 = \varepsilon$, and $l_n \in \mathcal{L}_f$. With a run we associate the timed word $(a_1, \delta_1)(a_2, \delta_1 + \delta_2) \ldots (a_n, \Sigma_{1 \leq i \leq n} \delta_i)$. The language accepted by the timed pushdown automaton $\mathcal{A}$, denoted by $L(\mathcal{A})$, is defined to be the set of timed words $w \in T\Sigma^+$ for which there exists a successful run of $\mathcal{A}$ that w is associated with.

Next we define some subclasses of timed pushdown automata; see Figure 1 for a graphical overview. We start with timed extensions of *one-counter automata* [18, 30] and *one-counter nets* [26, 1]. A *timed one-counter automaton* is a timed pushdown automaton where the stack alphabet is a singleton. By writing *push* and *pop* we mean that we increment and decrement the counter, respectively, whereas *empty*? corresponds to a zero test. A *timed one-counter net* is a timed one-counter automaton without zero tests, *i.e.*, the *empty*? operation is not allowed. We remark that for both classes, the execution of an edge of the form $(l, a, \phi, pop, \lambda, l')$ is *blocked* if the stack is empty.

Next, we consider the timed extension of an interesting subclass of pushdown automata called *visibly pushdown automata* [7]. A *timed visibly pushdown automaton* is a timed pushdown automaton for which the input alphabet Σ can be partitioned into three pairwise disjoint sets $\Sigma = \Sigma_{\mathsf{int}} \cup \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}}$ of *internal*, *call*, and *return* input symbols, respectively, and such that for every edge $(l, a, \phi, \mathsf{op}, \lambda, l')$ the following conditions are satisfied:

- $a \in \Sigma_{\mathsf{int}}$ if, and only if, $\mathsf{op} = noop$,
- $a \in \Sigma_{\mathsf{call}}$, if, and only if, $\mathsf{op} = push(b)$ for some $b \in \Gamma$,
- $a \in \Sigma_{\mathsf{ret}}$ if, and only if, $\mathsf{op} = empty?$ or $\mathsf{op} = pop(b)$ for some $b \in \Gamma$.

A *timed visibly one-counter automaton* (timed visibly one-counter net, respectively) is a timed one-counter automaton (timed one-counter net, respectively) that is also a timed visibly pushdown automaton. We say that a timed visibly one-counter net with no clocks is *deterministic* if for all $e = (l, a, \mathtt{true}, \mathsf{op}', \emptyset, l')$, $e' = (l, a, \mathtt{true}, \mathsf{op}'', \emptyset, l'') \in E$ with $e \neq e'$ we have either $\mathsf{op}' = pop$ and $\mathsf{op}'' = empty?$, or $\mathsf{op}' = empty?$ and $\mathsf{op}'' = pop$.

Finally, we define the class of *timed counter nets*, which generalizes timed one-counter nets, but is not a subclass of timed pushdown automata. A timed counter net of dimension

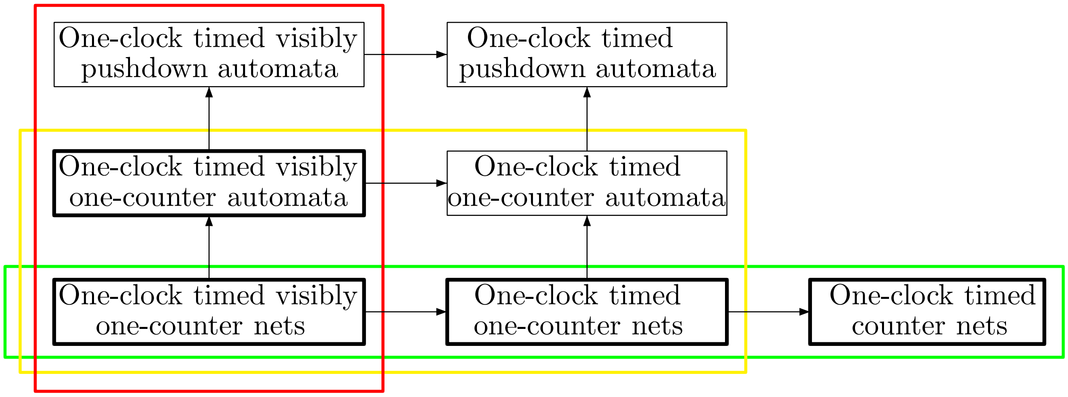
One-clock timed visibly pushdown automata
One-clock timed pushdown automata
One-clock timed visibly one-counter automata
One-clock timed one-counter automata
One-clock timed visibly one-counter nets
One-clock timed one-counter nets
One-clock timed counter nets

Figure 1: Extensions of one-clock timed automata with discrete data structures. Here, $\boxed{\text{A}}\!\to\!\boxed{\text{B}}$ means that A is a subclass of B. The classes surrounded by the red frame are visibly pushdown automata, in which the input determines the stack operations. The yellow framed classes only allow one counter, and the green framed classes do not allow zero tests. The language emptiness problem is decidable for all classes, but only the green framed classes have a decidable universality problem. The corresponding results for classes in boxes with bold line are new and presented in this paper.

n is a tuple $\mathcal{A} = (\Sigma, n, \mathcal{L}, \mathcal{L}_0, \mathcal{L}_f, \mathcal{X}, E)$, where $\mathcal{L}, \mathcal{L}_0, \mathcal{L}_f$ are the sets of locations, initial locations and accepting locations, respectively, and $E \subseteq \mathcal{L} \times \Sigma \times \Phi(\mathcal{X}) \times \{0, 1, -1\}^n \times 2^{\mathcal{X}} \times \mathcal{L}$ is a finite set of edges. A state of a timed counter net is a triple $(l, \nu, \vec{v})$, where $l \in \mathcal{L}$, ν is a clock valuation, and $\vec{v} \in \mathbb{N}^n$ is a vector representing the current values of the counters. We define $\langle (l, \nu, \vec{v}), \delta, a, (l', \nu', \vec{v'}) \rangle \in \Rightarrow_{\mathcal{A}}$ if, and only if, there exists some edge $(l, a, \phi, \vec{c}, \lambda, l') \in E$ such that $(\nu + \delta) \models \phi$, $\nu' = (\nu + \delta)[\lambda := 0]$, and $\vec{v'} = \vec{v} + \vec{c}$, where vector addition is defined pointwise. Note that, similar to pop operations on an empty stack, transitions which result in the negative value of one of the counters are *blocked*. The notions of *runs*, *successful runs*, *associated timed words* and *the language accepted by* $\mathcal{A}$, are defined analogously to the corresponding definitions for timed pushdown automata.

3. Main Results

In this section, we present the main results of the paper. We are interested in the language inclusion problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$, where $\mathcal{A}$ and $\mathcal{B}$ are extensions of timed automata with discrete data structures. Recall that according to standard notation in the field of verification, in this problem formulation $\mathcal{B}$ is seen as the *specification*, and $\mathcal{A}$ is the system that should satisfy this specification, *i.e.*, $\mathcal{A}$ should be a *model* of $\mathcal{B}$. As a special case of this problem, we consider the universality problem, *i.e.*, the question whether $L(\mathcal{B}) = T\Sigma^+$ for a given automaton $\mathcal{B}$. In general, the two problems are undecidable for timed pushdown automata. This follows on the one hand from the undecidability of the universality problem for timed automata [5], and on the other hand from the undecidability of the universality problem for pushdown automata. In fact, it is long known that the universality problem is undecidable already for non-deterministic one-counter automata [23, 28].

However, there are interesting decidability results for subclasses of timed pushdown automata: The language inclusion problem is decidable if $\mathcal{A}$ is a timed automaton, and $\mathcal{B}$ is a timed automaton with at most one clock [34]. As a special case, the universality problem for timed automata is decidable if only one clock is used. The language inclusion problem is also decidable if $\mathcal{A}$ is a one-counter net and $\mathcal{B}$ is a finite automaton, and if $\mathcal{A}$ is a finite automaton and $\mathcal{B}$ is a one-counter net [29]. The universality problem for non-deterministic one-counter nets has recently been proved to have non-primitive recursive complexity [27]. Further we know that the universality and language inclusion problems are decidable if $\mathcal{A}$ and $\mathcal{B}$ are visibly pushdown automata [7].

Hence it is interesting to consider the two problems for the corresponding subclasses of timed pushdown automata. It turns out that the decidability status changes depending on whether the *model* uses a stack (or, more detailed: a counter) or not. As a first main result we present:

Theorem 3.1. *The language inclusion problem is undecidable if $\mathcal{A}$ is a timed visibly one-counter net and $\mathcal{B}$ is a timed automaton, even if $\mathcal{A}$ is deterministic and has no clocks, and $\mathcal{B}$ uses at most one clock.*

We remark that this result corrects Theorem 2 in [20], in which it is claimed that the language inclusion problem for the case that $\mathcal{A}$ is a timed pushdown automaton and $\mathcal{B}$ is a one-clock timed automaton is decidable. The incorrectness of the proof of Theorem 2 in [20], respectively that of Theorem 1, which the proof for Theorem 2 builds upon, was already asserted in [15]. Since then the problem of whether language inclusion is decidable or not has been open.

In contrast to Theorem 3.1, we have decidability for the following classes:

Theorem 3.2. *The language inclusion problem is decidable with non-primitive recursive complexity if $\mathcal{A}$ is a timed automaton and $\mathcal{B}$ is a one-clock timed counter net.*

As a special case of this result (and with the lower bound implied by the corresponding result for one-clock timed automata [2]), we obtain:

Corollary 3.3. *The universality problem for one-clock timed counter nets is decidable with non-primitive recursive complexity.*

The next two sections are devoted to the proofs of Theorems 3.1 and 3.2. We will also give some interesting consequences of these results and their proofs. Amongst others, we prove the undecidability of the model checking problem for timed visibly one-counter nets and MTL over finite timed words, *cf.* the decidability of the same problem for timed automata [35]. After this, in Sect. 5, we will prove the following theorem:

Theorem 3.4. *The universality problem for one-clock timed visibly one-counter automata is undecidable.*

This is in contrast to the decidability of the universality problem for the two underlying models of one-clock timed automata [34] and visibly one-counter automata, which form a subclass of visibly pushdown automata [7]. We also want to point out that this result is stronger than a previous result on the undecidability of the universality problem for one-clock timed visibly pushdown automata (Theorem 3 in [20]), and our proof closes a gap in the proof of Theorem 3 in [20]. Further, we can infer from Corollary 3.3 and Theorem 3.4 the exact decidability border for the universality problem of timed pushdown automata, which lies between timed visibly one-counter nets and timed visibly one-counter automata.

4. Undecidability Results

In this section, we prove Theorem 3.1. The proof is a reduction of an undecidable problem for *channel machines*.

4.1. **Channel Machines.**

Let A be a finite alphabet. We define the order $\leq$ over the set of finite words over A by $a_1 a_2 \dots a_m \leq b_1 b_2 \dots b_n$ if there exists a strictly increasing function $f : \{1, \dots, m\} \to \{1, \dots, n\}$ such that $a_i = b_{f(i)}$ for every $i \in \{1, \dots, m\}$.

A *channel machine* consists of a finite-state automaton acting on an unbounded fifo channel. Formally, a channel machine is a tuple $\mathcal{C} = (S, s_I, M, \Delta)$, where

- S is a finite set of *control states*,
- $s_I \in S$ is the initial control state,
- M is a finite set of *messages*,
- $\Delta \subseteq S \times L \times S$ is the transition relation over the label set $L = \{!m, ?m \mid m \in M\} \cup \{\mathit{empty}?\}$.

Here, $!m$ corresponds to a *send* operation, $?m$ corresponds to a *read* operation, and *empty*? is a test which returns `true` if and only if the channel is empty. Without loss of generality, we assume that s_I does not have any incoming transitions, *i.e.*, $(s, l, s') \in \Delta$ implies $s' \neq s_I$. Further, we assume that $(s_I, l, s') \in \Delta$ implies $l = \mathit{empty}?$. A *configuration* of $\mathcal{C}$ is a pair (s, x), where $s \in S$ is the control state and $x \in M^*$ represents the contents of the channel. We use $\mathcal{H}^{\mathcal{C}}$ to denote the set of all configurations of $\mathcal{C}$. The rules in Δ induce a transition relation $\to_{\mathcal{C}}$ on $\mathcal{H}^{\mathcal{C}} \times L \times \mathcal{H}^{\mathcal{C}}$ as follows:

- $\langle (s, x), !m, (s', x') \rangle \in \to_{\mathcal{C}}$ if, and only if, there exists some transition $(s, !m, s') \in \Delta$ and $x' = x \cdot m$, *i.e.*, m is added to the tail of the channel.
- $\langle (s, x), ?m, (s', x') \rangle \in \to_{\mathcal{C}}$ if, and only if, there exists some transition $(s, ?m, s') \in \Delta$ and $x = m \cdot x'$, *i.e.*, m is removed from the head of the channel.
- $\langle (s, x), \mathit{empty}?, (s', x') \rangle \in \to_{\mathcal{C}}$ if, and only if, there exists some transition $(s, \mathit{empty}?, s') \in \Delta$ and $x = \varepsilon$, *i.e.*, the channel is empty, and $x' = x$.

We may write $(s, x) \xrightarrow{l}_{\mathcal{C}} (s', x')$ whenever $\langle (s, x), l, (s', x') \rangle \in \to_{\mathcal{C}}$. Next, we define a second transition relation $\rightsquigarrow_{\mathcal{C}}$ on $\mathcal{H}^{\mathcal{C}} \times L \times \mathcal{H}^{\mathcal{C}}$. The relation $\rightsquigarrow_{\mathcal{C}}$ is a superset of $\to_{\mathcal{C}}$. It contains some additional transitions which result from *insertion errors*. We define $\langle (s, x_1), l, (s', x_1') \rangle \in \rightsquigarrow_{\mathcal{C}}$, if, and only if, there exist $x, x' \in M^*$ such that $x_1 \leq x$, $\langle (s, x), l, (s', x') \rangle \in \to_{\mathcal{C}}$, and $x' \leq x_1'$. We may also write $(s, x) \overset{l}{\rightsquigarrow}_{\mathcal{C}} (s', x')$ whenever $\langle (s, x), l, (s', x') \rangle \in \rightsquigarrow_{\mathcal{C}}$. A *computation* of $\mathcal{C}$ is a finite sequence $\prod_{1 \leq i \leq k} \langle (s_{i-1}, x_{i-1}), l_i, (s_i, x_i) \rangle$ such that $\langle (s_{i-1}, x_{i-1}), l_i, (s_i, x_i) \rangle \in \rightsquigarrow_{\mathcal{C}}$ for every $i \in \{1, \dots, k\}$. We say that a computation is *error-free* if for all $i \in \{1, \dots, k\}$ we have $\langle (s_{i-1}, x_{i-1}), l_i, (s_i, x_i) \rangle \in \to_{\mathcal{C}}$. Otherwise, we say that the computation is *faulty*.

The proof of Theorem 3.1 is a reduction from the following undecidable [13] control state reachability problem: given a channel machine $\mathcal{C}$ with control states S and $s_F \in S$, does there exist an error-free computation of $\mathcal{C}$ from (s_I, ε) to (s_F, x) for some $x \in M^*$?

We remark that the analogous problem for faulty computations is decidable with non-primitive recursive complexity [2]: both the lower and upper bound can be proved using corresponding results for *lossy channel machines* [3, 40].

Example 4.1. *Define a channel machine* $\mathcal{C} = (S, M, s_I, \Delta)$ *by* $S = \{s_I, s, s', s_F\}$, $M = \{m_1, m_2, m_3\}$, *and* $\Delta = \{(s_I, \mathit{empty}?, s), (s, !m_1, s), (s, !m_2, s'), (s', ?m_1, s'), (s', ?m_3, s_F)\}$.

The computation

$$\gamma = (s_I, \varepsilon) \xrightarrow{empty?}_{\mathcal{C}} (s, \varepsilon) \xrightarrow{!m_1}_{\mathcal{C}} (s, m_1) \xrightarrow{!m_2}_{\mathcal{C}} (s', m_1 m_2) \overset{?m_1}{\rightsquigarrow}_{\mathcal{C}} (s', m_3 m_2) \xrightarrow{?m_3}_{\mathcal{C}} (s_F, m_2)$$

is faulty due to the last but one transition, where the symbol m_3 is an insertion error. It is easy to see that there exists no error-free computation from (s_I, ε) to (s_F, x) for some x.

The idea of our reduction is as follows: Given a channel machine $\mathcal{C}$, we define a timed language $L(\mathcal{C})$ consisting of all timed words that encode potentially faulty computations of $\mathcal{C}$ that start in (s_I, ε) and end in (s_F, x) for some $x \in M^*$. Then we define a timed visibly one-counter net $\mathcal{A}$ such that $L(\mathcal{A}) \cap L(\mathcal{C})$ contains exactly *error-free* encodings of such computations. In other words, we use $\mathcal{A}$ to exclude the encodings of faulty computations from $L(\mathcal{C})$, obtaining undecidability of the non-emptiness problem for $L(\mathcal{A}) \cap L(\mathcal{C})$. Finally, we define a one-clock timed automaton $\mathcal{B}$ that accepts the complement of $L(\mathcal{C})$; hence the problem of deciding whether $L(\mathcal{A}) \not\subseteq L(\mathcal{B})$ is undecidable.

4.2. **Encoding Faulty Computations.** For the remainder of Section 4, consider a channel machine $\mathcal{C} = (S, s_I, M, \Delta)$ and let $s_F \in S$. Define $\Sigma_{\mathsf{int}} := (S \backslash \{s_I\}) \cup M \cup L \cup \{\#\}$, $\Sigma_{\mathsf{call}} := \{s_I, +\}$, and $\Sigma_{\mathsf{ret}} := \{-, \star\}$, where $+, -, \#$ and $\star$ are fresh symbols that do not occur in $S \cup M \cup L$. The symbols $+, -, \#$ are called *wildcard symbols*. We define a timed language $L(\mathcal{C})$ over $\Sigma = \Sigma_{\mathsf{int}} \cup \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}}$ that consists of all timed words that encode computations of $\mathcal{C}$ from (s_I, ε) to (s_F, x) for some $x \in M^*$. The definition of $L(\mathcal{C})$ follows the ideas presented in [35], in which (a dual variant of) the control state reachability problem for channel machines is reduced to the satisfiability problem for MTL over timed words.

In general, the idea is to encode a configuration of $\mathcal{C}$ of the form (s, x) by a timed word of duration one. This timed word starts with the symbol s at some time t. If the content of the channel x is of the form $m_1 m_2 \dots m_j$, then s is followed by the symbols $m_1, m_2, \dots, m_j$ in this order. The timestamps of these symbols must be strictly monotonic and in the interval $(t, t+1)$. Due to the denseness of the time domain, one can indeed store the channel content in one time unit without any upper bound on j.

Example 4.2. *The initial configuration (s, ε) is encoded by a single-letter timed word, for instance by $(s, 1.0)$. The configuration $(s', m_1 m_2)$ may be encoded by the timed word $(s', 7.0)(m_1, 7.2)(m_2, 7.8)$. The choice of the timestamps is arbitrary as long as the timestamps of the message symbols are in the unit interval determined by the timestamp of the preceding control state.*

To encode a computation of a channel machine, we concatenate the encodings of each of the participating configurations in the following way. Encodings of consecutive configuration have a time distance of exactly two time units. One time unit before the encoding of the next configuration we store the label of the transition. Further, each message symbol in the encoding of the current configuration has a matching copy in the encoding of the next configuration, after exactly two time units, and in accordance with the following rules: If the next transition is sending a new message symbol m to the tail of the channel, we add m to the tail of the encoding of the next configuration. If the next transition is reading m from the head of the channel, we test whether the first symbol after the control state in the encoding of the current configuration equals m, and if so, we remove it from the encoding of the next configuration.

Example 4.3. *The transition* $(s', m_1 m_2) \xrightarrow{?m_1}_{\mathcal{C}} (s', m_2)$ *may be encoded by*

$$(s', 7.0)(m_1, 7.2)(m_2, 7.8)(?m_1, 8.0)(s', 9.0)(m_2, 9.8).$$

The symbol m_2 *at time* 7.8 *has a matching copy exactly two time units later; the symbol* m_1 *at* 7.2 *does not, because it is removed from the channel by the transition.*

This idea of encoding computations of channel machines was used for proving lower complexity bounds for the satisfiability problem of MTL [35] and the universality problem for one-clock timed automata [2]. These problems, however, are decidable [35, 2], whereas here we want to use the encoding to show undecidability of a problem. The crucial point is that the encoding explained above does not exclude timed words that are encoding *faulty* computations of a channel machine: for excluding encodings of faulty computations, we need to require that every message symbol has not only a matching copy *after* two time units, but also a matching copy two time units *before*. In other words, it should not be possible that message symbols appear "all of a sudden", *i.e.*, without a corresponding error-free transition.

Example 4.4. *The faulty transition* $(s', m_1 m_2) \overset{?m_1}{\rightsquigarrow}_{\mathcal{C}} (s', m_3 m_2)$ *may be encoded by*

$$(s', 7.0)(m_1, 7.2)(m_2, 7.8)(?m_1, 8.0)(s', 9.0)(m_3, 9.1)(m_2, 9.8).$$

As in the previous example, the symbol m_2 *at time* 7.8 *has a matching copy exactly two time units later; the symbol* m_1 *at* 7.2 *does not, because it is removed from the channel by the encoded transition. However, the symbol* m_3 *at time* 9.1 *appears all of a sudden, i.e., without any matching copy two time units before. This corresponds to an insertion error in the computation, see Example 4.1.*

The above described *backward-looking conditions*, however, cannot be expressed by neither MTL formulas, nor by one-clock timed automata.[1] Due to this failure, it is only the control state reachability problem *for faulty computations* that can be reduced to the satisfiability problem for MTL respectively the universality problem for one-clock timed automata. As mentioned before, the control state reachability problem for faulty computations is *decidable* [2].

For our undecidability proof to work, we have to exclude encodings of faulty computations. In other words, we have to exclude timed words in which message symbols occur without any matching copy two time units before. This will be carried out by the counter of the visibly timed one-counter net $\mathcal{A}$. For this to work, we have to change the encoding in some details, as explained in the following.

Assume we want to encode a given error-free computation of $\mathcal{C}$. Let n be the maximum length of the channel content during this computation. Let us assume for a moment that $\mathcal{C}$ does not start its computation with the empty channel, but the channel contains the word $\#^n$, *i.e.*, n occurrences of the wildcard symbol $\#$. The semantics of $\mathcal{C}$ is changed in the following way: If a message m is sent, then, instead of adding m to the tail of the channel, the first # occurring in the channel is *replaced* by m. If a message m is read, then, it is tested whether the first symbol in the channel is m. If this the case, it is removed from the channel and additionally a new wildcard symbol # is added to the tail of the channel. A

[1] Backward-looking conditions (or, to be more exact with respect to the reduction: the violation of such backward-looking conditions), can be expressed by MTL with past operators, and by timed automata with two clocks, leading to the undecidability of the corresponding satisfiability and universality problem, respectively [5].

test for emptiness of the channel is replaced by testing whether the channel only contains the wildcard symbol #.

Example 4.5. *Let* $(s, m_1m_1) \xrightarrow{!m_2}_{\mathcal{C}} (s', m_1m_1m_2) \xrightarrow{?m_1}_{\mathcal{C}} (s', m_1m_2)$ *be a computation of* $\mathcal{C}$*, and let* $n = 4$*. With the new wildcard semantics this computation is of the form*

$$(s, m_1m_1\#\#) \xrightarrow{!m_2}_{\mathcal{C}} (s', m_1m_1m_2\#) \xrightarrow{?m_1}_{\mathcal{C}} (s', m_1m_2\#\#).$$

Observe that the length of the channel content is constantly n. We will later exploit this fact.

The encoding of computations of channel machines is now changed with this wildcard semantics in mind. The initial configuration is encoded by a timed word that starts with the symbol s_I at some time t, and then is followed by n occurrences of #, all of which have monotonically increasing timestamps in the interval $(t, t + 1)$. The rules for the transitions change accordingly: If the next transition is sending a message m to the channel, we *replace* in the encoding of the next configuration the first occurrence of # by m. If the next transition is reading a message m from the head of the channel, we test whether the first symbol after the control state is m, and if so, we remove it from the encoding of the next configuration. We further add a new # to the end of the encoding of the next configuration.

Example 4.6. *The encoding of the computation of the previous example may be of the form*

$$(s, 7.0)(m_1, 7.2)(m_1, 7.5)(\#, 7.6)(\#, 7.8)(!m_2, 8.0)(s', 9.0)(m_1, 9.2)(m_1, 9.5)(m_2, 9.6)(\#, 9.8)$$
$$(?m_1, 10.0)(s', 11.0)(m_1, 11.5)(m_2, 11.6)(\#, 11.8)(\#, 11.9).$$

Observe that the length of the encoding of every configuration is constantly $n+1$. This is due to the fact that none of the encoding rules changes the number of symbols in a configuration. However, due to the lack of backward-looking conditions, it may still happen that some symbols appear "all of a sudden", *i.e.*, without a matching copy two time units before. In this case, the length of the encoding of the configuration increases. By the encoding conditions above, the increasing effect will be carried over to the encodings of the following configurations. Hence, in order to find out whether insertion errors occurred, it suffices to compare the length of the encoding of the initial configuration (which is equal to $n + 1$) to the length of the encoding of the last configuration: If it is still $n + 1$, then we know that no insertion error has occurred; otherwise, some insertion error has occurred. The test will be done by the counter of the timed visibly one-counter net $\mathcal{A}$. During a run of $\mathcal{A}$ on the encoding of a computation, the counter is incremented while the symbols of the encoding of the first configuration are read, and it is decremented while the symbols of the encoding of the last configuration are read. In between, the counter is not touched. By the fact that a one-counter net cannot decrement the counter more often than it was incremented ($\mathcal{A}$ is blocked as soon as the counter would become negative), we can exclude timed words which are encoding potential insertion errors.

By definition of timed *visibly* one-counter nets, $\mathcal{A}$ is restricted to use s_I and the wildcard symbol $+$ whenever the counter should be incremented, and it can only use the wildcard symbol $-$ and $\star$ whenever the counter should be decremented. This requires some extra effort in the encoding, namely that the encoding of the first configuration only uses the wildcard symbol $+$, and the encoding of the last configuration only uses the wildcard symbol $-$. Before we give the formal definition of the language $L(\mathcal{C})$, we show a complete encoding of the faulty computation of Example 4.1 for $n = 2$.

Example 4.7.

$$(s_I,1.0)(+,1.2)(+,1.8)(\mathit{empty}?,2.0)(s,3.0)(\#,3.2)(\#,3.8)(!m_1,4.0)(s,5.0)(m_1,5.2)(\#,5.8)$$
$$(!m_2,6.0)(s',7.0)(m_1,7.2)(m_2,7.8)(?m_1,8.0)(s',9.0)(m_3,9.1)(m_2,9.8)(\#,9.9)(?m_3,10.0)$$
$$(s_F,11.0)(-,11.8)(-,11.9)(-,11.95)(\star,12.0)$$

For every $n \in \mathbb{N}$, we define a timed language $L(\mathcal{C},n)$ as follows: The timed language $L(\mathcal{C},n)$ consists of all timed words w over Σ that satisfy the following conditions:

(1) w must be strictly monotonic.
(2) The untiming of w must be of the form given by the following regular expression:
$$s_I(+)^n \mathit{empty}?(SM^*\#^*L)^* s_F -^* \star$$
(3) For every $s \in S$, if s is followed by l after one time unit, and s is followed by s' after two time units, then $(s,l,s') \in \Delta$.
(4) For every $s \in S$ with $s \neq s_F$, there exists $l \in L$ after exactly one time unit.
(5) For every $s \in S$ with $s \neq s_F$, there exists $s' \in S$ after exactly two time units.
(6) After s_F the symbol $\star$ occurs after two time units.

Further, for every infix of w of the form
$$(s,\delta)(\sigma_1,\delta_1)(\sigma_2,\delta_2)\dots(\sigma_k,\delta_k)(l,\delta+1)(s',\delta+2)(\sigma'_1,\delta'_1)\dots(\sigma'_{k'},\delta'_{k'})(l',\delta+3)$$
with $s,s' \in S\backslash\{s_F\}$, $l,l' \in L$, $\delta \in \mathbb{R}_{\geq 0}$, $\delta < \delta_1 < \cdots < \delta_k < \delta+1 < \delta+2 < \delta'_1 < \cdots < \delta'_{k'} < \delta+3$, there exists a strictly increasing function $f : \{1,\dots,k\} \to \{1,\dots,k'\}$ such that the following conditions are satisfied:

(7) If $l = \mathit{empty}?$, then
   (a) $\sigma_i \in \{+,\#\}$ and $\sigma'_{f(i)} = \#$ for all $i \in \{1,\dots,k\}$ (the channel is empty), and
   (b) $\delta'_{f(i)} = \delta_i + 2$ for every $i \in \{1,\dots,k\}$ (there are matching copies after two time units).
(8) If $l = !m$ for some $m \in M$, then
   (a) there exists $i \in \{1,\dots,k\}$ such that $\sigma_i = \#$ (there is some wildcard symbol in the encoding of the current configuration),
   (b) $\sigma'_{f(j)} = m$ and $\delta'_{f(j)} = \delta_j + 2$ for $j = \min\{i \in \{1,\dots,k\} \mid \sigma_i = \#\}$ (the first wildcard symbol is replaced by m two time units later),
   (c) $\sigma'_{f(i)} = \sigma_i$ and $\delta'_{f(i)} = \delta_i + 2$ for all $i \in \{1,\dots,k\}\backslash\{j\}$ (there are matching copies for the remaining symbols).
(9) If $l = ?m$ for some $m \in M$, then
   (a) $\sigma_1 = m$ (the first symbol is equal to m),
   (b) $f(k) = k'$, $\sigma'_{f(k)} = \sigma_{k'} = \#$, and $\delta'_{f(k)} = \delta'_{k'} > 2 + \delta_k$ (a new wildcard symbol is added at the end of the encoding), and
   (c) $\sigma'_{f(i)} = \sigma_{i+1}$ and $\delta'_{f(i)} = 2 + \delta_{i+1}$ for all $i \in \{1,\dots,k-1\}$ (there are matching copies for all other symbols except for the first one, which is removed from the encoding).

For an infix like above but with $s' = s_F$ and $l' = \star$, we add conditions (7'), (8') and (9') that differ from 7, 8, and 9, respectively, in that all message or wildcard symbols to be copied or added to the encoding of the next configuration are replaced by the wildcard symbol $-$.

A simple observation that we will later use is that by these conditions the length of the encodings of two consecutive configurations cannot decrease. Indeed, the conditions for representing transitions between two configurations of the channel machine do not change

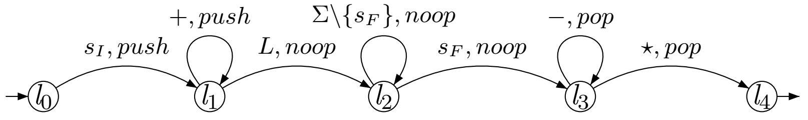

Figure 2: The deterministic timed visibly one-counter net $\mathcal{A}$ for excluding insertion errors.

the number of symbols in the encoding of the respective configurations. By the lack of backward-looking conditions it may however happen that some symbols appear all of a sudden, *i.e.*, without a matching copy two time units before. We point out that such insertion errors *may* occur, but they are not required by any of the conditions.

4.3. **Excluding Faulty Computations.** We define a timed visibly one-counter net $\mathcal{A}$ over Σ such that for every $n \in \mathbb{N}$ the intersection $L(\mathcal{A}) \cap L(\mathcal{C}, n)$ consists of all timed words that encode *error-free* computations of $\mathcal{C}$ from (s_I, ε) to (s_F, x) for some $x \in M^*$. The timed visibly one-counter net $\mathcal{A}$ is shown in Figure 1. After incrementing the counter while reading the initial letter s_I, it non-deterministically guesses a number $n \in \mathbb{N}$ of symbols $+$ and increments the counter each time it reads the symbol $+$. When $\mathcal{A}$ leaves l_1, the value of the counter is $n + 1$. After that, the counter value is not changed until the state symbol s_F is read. Then, while reading symbols in $\{-, \star\}$, the counter value is decremented. Note that $\mathcal{A}$ can reach the final location l_4 only if the number of the occurrences of symbol $-$ between s_F and $\star$ is *at most* n: otherwise the counter value would become negative, and thus the edges going out from l_3 would be blocked. Note that $\mathcal{A}$ does not use any clock, and it is deterministic.

Example 4.8. *The timed word presented in Example 4.7 cannot be accepted by the timed visibly one-counter net $\mathcal{A}$ in Figure 2: The run of $\mathcal{A}$ on the prefix is of the form*

$$\langle (l_0, 0), 1.0, s_I, (l_1, 1)\rangle\langle (l_1, 1), 0.2, +, (l_1, 2)\rangle\langle (l_1, 2), 0.6, +, (l_1, 3)\rangle\langle (l_1, 3), 0.2, \mathit{empty?}, (l_2, 3)\rangle.$$

Here, in (l, c), l stands for the current location, and c stands for the current value of the counter. The counter value stays constant until we finally read the first $-$:

$$\langle (l_2, 3), 1.0, s_F, (l_3, 3)\rangle\langle (l_3, 3), 0.8, -, (l_3, 2)\rangle\langle (l_3, 2), 0.1, -, (l_3, 1)\rangle\langle (l_3, 1), 0.05, -, (l_3, 0)\rangle.$$

Now $\mathcal{A}$ is blocked: all outgoing edges of l_3 require the counter to be decremented, which is not possible if the counter has value zero. This is indeed what we want: the timed word in Example 4.1 is encoding a faulty computation, and should be excluded.

Lemma 4.9. *$\mathcal{C}$ has an error-free computation from (s_I, ε) to (s_F, x) for some $x \in M^*$, if, and only if, there exists $n \in \mathbb{N}$ such that $L(\mathcal{C}, n) \cap L(\mathcal{A}) \neq \emptyset$.*

Proof. For the direction from left to right, let γ be an error-free computation of $\mathcal{C}$ from (s_I, ε) to (s_F, x) for some $x \in M^*$. Let n be the maximum length of the channel content during γ. Let w be a timed word in $L(\mathcal{C}, n)$ in which no message or wildcard symbols occur "all of a sudden", *i.e.*, without a matching copy two time units before. Note that such a timed word exists, because γ is error-free, and hence there is no need to encode insertion errors into w. This implies that the length of the encodings of all, and in particular, the first and the last configuration in w is $n + 1$. This implies $w \in L(\mathcal{A})$. Hence $L(\mathcal{C}, n) \cap L(\mathcal{A}) \neq \emptyset$.

For the direction from right to left, let $w \in L(\mathcal{C}, n) \cap L(\mathcal{A})$ for some $n \in \mathbb{N}$. By definition of $L(\mathcal{C}, n)$, the length of the encoding of the initial configuration is $n + 1$. By the observation above, the length of the encoding of the last configuration is thus at least $n+1$, too. However, by definition of $\mathcal{A}$, the length of the encoding of the last configuration cannot be greater than $n+1$, because otherwise the edge to l_4 cannot be taken due to the decrement operation. By the observation above, the length of encodings of consecutive configurations do not decrease, and thus the length of the encodings of all configurations is $n+1$. Hence we can conclude that there are no insertion errors necessary to encode the execution of a transition. This implies that there exists some error-free computation γ of $\mathcal{C}$ from (s_I, ε) to (s_F, x) for some $x \in M^*$. □

We finally define $L(\mathcal{C}) := \bigcup_{n \in \mathbb{N}} L(\mathcal{C}, n)$.

Corollary 4.10. *There exists some error-free computation of $\mathcal{C}$ from (s_I, ε) to (s_F, x) for some $x \in M^*$ if, and only if, $L(\mathcal{A}) \cap L(\mathcal{C}) \neq \emptyset$.*

4.4. **The Reduction.** Finally, we define a one-clock timed automaton $\mathcal{B}$ such that $L(\mathcal{B}) = T\Sigma^+ \backslash L(\mathcal{C})$. The construction of $\mathcal{B}$ follows the same ideas as, *e.g.*, in [4]: $\mathcal{B}$ is the union of several one-clock timed automata, each of them violating some condition of the definition of $L(\mathcal{C})$, as described in the following.

The timed automaton in Figure A accepts timed words that are not strictly monotonic, thus violating condition (1). For accepting timed words violating condition (2), we can construct a finite automaton that recognizes the complement of the given regular expression. Define for every $s \in S\backslash\{s_f\}$ the sets $\overline{L}(s) = \{l \in L \mid \neg\exists s' \in S.(s, l, s') \in \Delta\}$ and $\overline{S}(s) = \{s' \in S \mid \neg\exists l \in L.(s, l, s') \in \Delta\}$, and $\overline{L}(s_F) = \Sigma\backslash\{\star\}$ and $\overline{S}(s_F) = \Sigma$. Then for every s the corresponding timed automaton in Figure B accepts timed words that contain the encoding of a transition $(s, l, s') \notin \Delta$, thus violating condition (3). Violations of the forward-looking conditions in (4), (5), and (6) can be accepted by the timed automaton in Figure C with, respectively, (4) $M_1 = S$, $M_2 = L$, and $k = 1$, (5) $M_1 = S$, $M_2 = S$, and $k = 2$, and (6) $M_1 = \{s_F\}$, $M_2 = \{\star\}$, and $k = 1$.

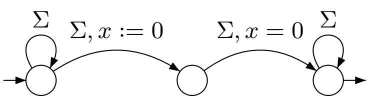

Figure A: Condition (1)

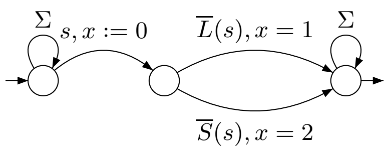

Figure B: Condition (3)

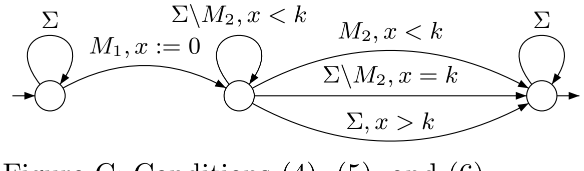

Figure C: Conditions (4), (5), and (6)

Timed words violating condition (7a) can be accepted by the timed automaton in Figure D. In Figure E we show a timed automaton that accepts timed words violating condition (7b).

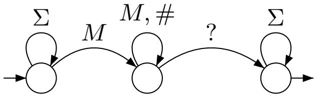

Figure D: Condition (7a)

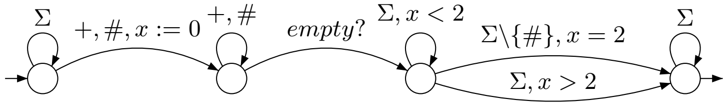

Figure E: Condition (7b)

The timed automaton in Figure F accepts all timed words for which the last symbol before $!m$ is different from $\#$. This, together with the structure of w ensured by condition (2), implies that condition (8a) is violated. The timed automaton in Figure G accepts timed words violating condition (8b). For condition (8c), we can use a timed automaton similar to that in Figure E. By constructing for every $m \in M$ the corresponding automata, we can thus accept all timed words violating the conditions stated in (8).

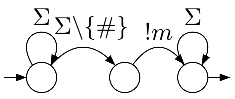

Figure F: Condition (8a)

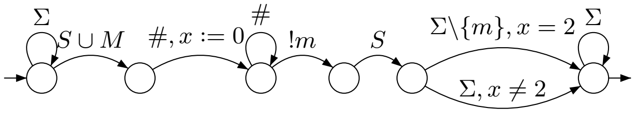

Figure G: Condition (8b)

Last but not least we present timed automata that accept timed words violating condition (9). For all $m \in M$, we define timed automata shown in Figure H und I, respectively, that accept timed words violating condition (9a) and (9b), respectively. For (9c) we construct a timed automaton very similar to that in Figure E.

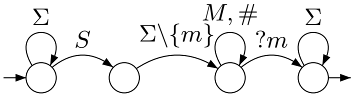

Figure H: Condition (9a)

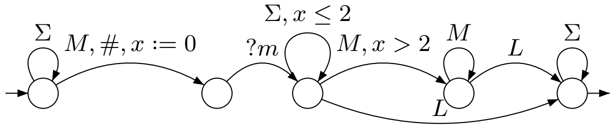

Figure I: Condition (9b)

So let $\mathcal{B}$ the union of all these timed automata. One can easily see that for every timed word $w \in T\Sigma^+$ we have $w \in L(\mathcal{B})$ if, and only if, $w \notin L(\mathcal{C})$. By Corollary 4.10, there exists some error-free computation of $\mathcal{C}$ from (s_I, ε) to (s_F, x) for some $x \in M^*$ if, and only if, $L(\mathcal{A}) \cap L(\mathcal{C}) \neq \emptyset$. The latter is equivalent to $L(\mathcal{A}) \not\subseteq L(\mathcal{B})$. Hence, the language inclusion problem is undecidable.

4.5. **Undecidability of the Model Checking Problem for MTL.** The proof idea of Theorem 3.1 can be used to show the undecidability of the following model checking problem: given a timed visibly one-counter net $\mathcal{A}$, and an MTL formula φ, does every $w \in L(\mathcal{A})$ satisfy φ? Recall that this problem is decidable for the class of timed automata [35]. We prove that adding a visibly counter without zero test already makes the problem undecidable. But first, let us recall the syntax and semantics of MTL.

Let Σ be a finite alphabet. The set of MTL formulae is built up from Σ by Boolean connectives and a time constraining version of the *until* modality:

$$\varphi ::= \texttt{true} \mid a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathsf{U}_I \varphi_2$$

where $a \in \Sigma$ and $I \subseteq \mathbb{R}_{\geq 0}$ is an open, closed, or half-open interval with endpoints in $\mathbb{N} \cup \{\infty\}$.

We interpret MTL formulae in the *pointwise semantics*, *i.e.*, over finite timed words over Σ. Let $w = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$ be a timed word, and let $i \in \{1, \dots, n\}$. We define the *satisfaction relation for MTL*, denoted by $\models$, inductively as follows:

$$\begin{aligned}
(w,i) \models a \quad &\Leftrightarrow \quad a_i = a \\
(w,i) \models \neg\varphi \quad &\Leftrightarrow \quad (w,i) \not\models \varphi, \\
(w,i) \models \varphi_1 \wedge \varphi_2 \quad &\Leftrightarrow \quad (w,i) \models \varphi_1 \text{ and } (w,i) \models \varphi_2, \\
(w,i) \models \varphi_1 \mathsf{U}_I \varphi_2 \quad &\Leftrightarrow \quad \exists j. i < j \leq n : (w,j) \models \varphi_2 \text{ and } t_j - t_i \in I, \\
&\qquad \text{and } \forall k. i < k < j : (w,k) \models \varphi_1.
\end{aligned}$$

We say that a timed word $w \in T\Sigma^+$ satisfies an MTL formula φ, written $w \models \varphi$, if $(w, 1) \models \varphi$.

Note that MTL only allows to express restrictions on time, and it does not allow for any restrictions on the values of the counters. In fact, it is proved that as soon as we add to MTL the capability for expressing restrictions on the values of a counter that can be incremented and decremented, model checking is undecidable [38]. The proof of the following theorem is based on the fact that MTL can encode computations of channel machines with insertion errors [35].

Theorem 4.11. *The model checking problem for timed visibly one-counter nets and MTL is undecidable, even if the timed visibly one-counter net does not use any clocks and is deterministic.*

Proof. The definition of an MTL formula φ such that $L(\varphi) = L(\mathcal{C})$ is straightforward, see, *e.g.*, [35]. By Corollary 4.10, there exists some error-free computation of $\mathcal{C}$ from (s_I, ε) to (s_F, x) for some $x \in M^*$ if, and only if, $L(\mathcal{A}) \cap L(\varphi) \neq \emptyset$. The latter, however, is equivalent to saying that there exists some timed word $w \in L(\mathcal{A})$ such that $w \not\models \varphi$. Hence the model checking problem is undecidable. □

We would like to remark that the proof of Theorem 4.11 shares some similarities with the proof of the undecidability of model checking one-counter machines (*i.e.*, one-counter automata without input alphabet) and Freeze LTL with one register ($\mathrm{LTL}_1^{\downarrow}$, for short) [18]. In [17], it is proved that $\mathrm{LTL}_1^{\downarrow}$ can encode computations of *counter automata with incrementing errors*. Similar to the situation for MTL and channel machines, $\mathsf{LTL}_1^{\downarrow}$ can however not encode *error-free* computations of counter automata. In [18], a one-counter machine is used to repair this incapability, resulting in the undecidability of the model checking problem. The one-counter machine in [18] does not use zero tests; however, we point out that in contrast to our visibly timed one-counter net the one-counter machine in [18] is *non-deterministic*. Indeed, model checking *deterministic* one-counter machines and $\mathrm{LTL}_1^{\downarrow}$ is decidable [18].

We further remark that using a similar proof, we can show that the model checking problem for *parametric timed automata* and MTL is undecidable, even if the automaton only uses one parametric clock, one parameter and is deterministic [39].

4.6. **Energy Problems on Timed Automata with Discrete Weights.** Next we will consider an interesting extension of *lower-bound energy problems on weighted timed automata*, introduced in [11], which gained attention in the last years, see, *e.g.*, [12, 37, 10]. In lower-bound energy problems, one is interested whether in a given automaton with some weight variable whose value can be increased and decreased, there exists a successful run in which all accumulated weight values are never below zero. Similar problems have also been considered for untimed settings, *e.g.*, [31, 21, 22, 14].

A *timed automaton with discrete weights* (dWTA, for short) is syntactically the same as a timed one-counter net. In the semantical graph induced by a dWTA, however, we allow the value of the counter (or, the *weight variable*) to become negative. Hence the value of the weight variable does not influence the behaviour of the dWTA, because, different to timed one-counter nets, transitions that result in negative values are not blocked. We remark that for the simple reasons that (1) the value of the weight variable does not influence the behaviour of dWTA, and (2) MTL does not restrict the values of the weight variable, the model checking problem for dWTA and MTL is decidable, using the same algorithm as for timed automata [35]. We define the *model checking energy problem* for dWTA and MTL as follows: given a dWTA $\mathcal{A}$ and an MTL formula φ, does there exist some accepting run ρ of $\mathcal{A}$ such that the value of the weight variable is always non-negative, and the timed word w associated with ρ satisfies φ? For the special case $\varphi = \texttt{true}$, the problem is decidable in polynomial time for one-clock dWTA [11].

Theorem 4.12. *The model checking energy problem for dWTA and MTL is undecidable, even if the dWTA uses no clocks.*

Proof. For the proof, we reduce the model checking problem for timed one-counter nets and MTL to the energy problem. Note that timed one-counter nets are a generalization of timed visibly one-counter nets, and thus by Theorem 4.11 the model checking problem is undecidable. Let $\mathcal{A}$ be a timed one-counter net, and let φ be an MTL formula. Define $\mathcal{A}'$ to be the dWTA that is syntactically equal to $\mathcal{A}$. One can easily prove that $(\mathcal{A}, \varphi)$ is a negative instance of the model checking problem if, and only if, $(\mathcal{A}', \neg\varphi)$ is a positive instance of the energy problem. □

5. Decidability Result

In the preceding section, we showed that one cannot automatically verify timed automata extended with unbounded discrete data structures against real-time specifications expressed by timed automata or MTL-formulas. In this section, we prove that in contrast to this, we can use one-clock timed counter nets as specification for model checking timed automata: the language inclusion problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is decidable with non-primitive recursive complexity if $\mathcal{A}$ is a timed automaton and $\mathcal{B}$ is a one-clock timed counter net (Theorem 3.2). We will first give the formal proof of Theorem 3.2. After that, we will argue that this result extends known facilities for the verification of timed automata.

5.1. Proof of Theorem 3.2.

Proof. For the case that $\mathcal{A}$ is a timed automaton and $\mathcal{B}$ is a one-clock timed automaton, the language inclusion problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is decidable [34] with non-primitive recursive complexity [2]. The lower bound hence follows, and the decidability proof is an adaptation of the decidability proof in [34].

The proof is based on the theory of well-quasi-orders, and we start with defining some useful notions.

Let A, B be two sets, and let $\preceq$ be a binary relation on A. Then $\preceq$ is a *quasi-order* on A if $\preceq$ is reflexive and transitive. $\preceq$ is a *well-quasi-order* on A if it is a quasi-order and for every infinite sequence $a_1, a_2, a_3, \ldots$ in A there exist indices $i < j$ such that $a_i \preceq a_j$. A standard example for a well-quasi-order is the pointwise order $\leq^k$ on the set $\mathbb{N}^k$ of vectors of k natural numbers (Dickson's Lemma, [19]).

Let $\preceq$ be a quasi-order on A, and let $\sqsubseteq$ be a quasi-order on B. We define the *product* of $\preceq$ and $\sqsubseteq$ on $(A \times B)$ by $(a, b) \leq (a', b')$, if and only if, $a \preceq a'$ and $b \sqsubseteq b'$. We define the *monotone domination order* $\preceq^*$ on A^* by $a_1 a_2 \ldots a_m \preceq^* a'_1 a'_2 \ldots a'_n$ if and only if there exists a strictly increasing function $f : \{1, \ldots, m\} \to \{1, \ldots, n\}$ such that for all $i \in \{1, \ldots, m\}$ we have $a_i \preceq a'_{f(i)}$. We define the *subset order* $\preceq^{\mathcal{P}}$ on the set $\mathcal{P}(A)$ of finite subsets of A by $A_1 \preceq^{\mathcal{P}} A_2$ if and only if there is an injective mapping $f : A_1 \to A_2$ such that for all $a \in A_1$ we have $a \preceq f(a)$.

Lemma 5.1 (Higman's Lemma [25])**.** (1) *If $\preceq$ and $\sqsubseteq$ are well-quasi-orders on A and B, respectively, then the product of $\preceq$ and $\sqsubseteq$ is a well-quasi-order on $(A \times B)$.*

(2) *If $\preceq$ is a well-quasi-order on A, then the monotone domination order $\preceq^*$ is a well-quasi-order on A^*.*

(3) *If $\preceq$ is a well-quasi-order on A, then the subset order $\preceq^{\mathcal{P}}$ is a well-quasi-order on $\mathcal{P}(A)$.*

Let $\mathcal{A} = (\Sigma^{\mathcal{A}}, \Gamma^{\mathcal{A}}, \mathcal{L}^{\mathcal{A}}, \mathcal{L}_0^{\mathcal{A}}, \mathcal{L}_f^{\mathcal{A}}, \mathcal{X}, E^{\mathcal{A}})$ be a timed automaton with clock variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, and let $\mathcal{B} = (\Sigma^{\mathcal{B}}, \mathfrak{n}, \mathcal{L}^{\mathcal{B}}, \mathcal{L}_0^{\mathcal{B}}, \mathcal{L}_f^{\mathcal{B}}, \{x\}, E^{\mathcal{B}})$ be a timed counter net of dimension $\mathfrak{n}$ with a single clock variable x. Without loss of generality, we may assume that $\Sigma^{\mathcal{A}} = \Sigma^{\mathcal{B}}$.

Note that a state (l, ν) of $\mathcal{A}$ is an element in $\mathcal{L}^{\mathcal{A}} \times (\mathbb{R}_{\geq 0})^{\mathcal{X}}$, and a state $(l, v, \vec{u})$ of $\mathcal{B}$ is an element in $\mathcal{L}^{\mathcal{B}} \times \mathbb{R}_{\geq 0} \times \mathbb{N}^{\mathfrak{n}}$. A *joint configuration of $\mathcal{A}$ and $\mathcal{B}$* is a pair (q, γ), where q is a state of $\mathcal{A}$, and γ is a set of states of $\mathcal{B}$. We use $\mathcal{G}^{\mathcal{AB}}$ to denote the set of all joint configurations of $\mathcal{A}$ and $\mathcal{B}$. We say that a joint configuration (q, γ) is *initial* if $q \in (\mathcal{L}_0^{\mathcal{A}} \times \{0\}^{\mathcal{X}})$ and $\gamma = \{(l, 0, \vec{0}) \mid l \in \mathcal{L}_0^{\mathcal{B}}\}$ (with $\vec{0}$ we denote the vector of dimension $\mathfrak{n}$ containing only 0). We say that a joint configuration (q, γ) is *bad* if $q = (l, \nu)$ for some $l \in \mathcal{L}_f^{\mathcal{A}}$, and for all states $(l', v', \vec{u}') \in \gamma$ we have $l' \notin \mathcal{L}_f^{\mathcal{B}}$. The *joint behaviour of $\mathcal{A}$ and $\mathcal{B}$* is defined as follows: For a state (l, ν) of $\mathcal{A}$, $\delta \in \mathbb{R}_{\geq 0}$ and $a \in \Sigma$, we define $\mathsf{Succ}^{\mathcal{A}}((l, \nu), \delta, a) = \{(l', \nu') \mid \langle (l, \nu), \delta, a, (l', \nu') \rangle \in \Rightarrow_{\mathcal{A}}\}$. For a set γ of states of $\mathcal{B}$, we define $\mathsf{Succ}^{\mathcal{B}}(\gamma, \delta, a) = \{(l', v', \vec{u}') \mid \exists (l, v, \vec{u}) \in \gamma . \langle (l, v, \vec{u}), \delta, a, (l', v', \vec{u}') \rangle \in \Rightarrow_{\mathcal{B}}\}$. Note that $\mathsf{Succ}^{\mathcal{B}}(\gamma, \delta, a)$ is a set of states of $\mathcal{B}$, and it may be empty. Finally, we define the transition relation $\Rightarrow_{\mathcal{AB}}$ on $\mathcal{G}^{\mathcal{AB}} \times \Sigma \times \mathcal{G}^{\mathcal{AB}}$ by $\langle (q, \gamma), a, (q', \gamma') \rangle \in \Rightarrow_{\mathcal{AB}}$ if there exists some $\delta \in \mathbb{R}_{\geq 0}$ such that $q' \in \mathsf{Succ}^{\mathcal{A}}(q, \delta, a)$ and $\gamma' = \mathsf{Succ}^{\mathcal{B}}(\gamma, \delta, a)$.

Next, we encode joint configurations of $\mathcal{A}$ and $\mathcal{B}$ by finite untimed words over the set Λ of finite subsets of $(\mathcal{L}^{\mathcal{A}} \times \mathcal{X} \times \mathsf{reg} \times \vec{0}) \cup \{\mathcal{L}^{\mathcal{B}} \times \{x\} \times \mathsf{reg} \times \mathbb{N}^{\mathfrak{n}})$. Here, $\mathsf{reg} := \{0, 1, \ldots, \mathsf{cmax}\} \cup \{\top\}$, where cmax is an integer greater than the maximal constant occurring in clock constraints in both $\mathcal{A}$ and $\mathcal{B}$, and $\top$ is a symbol representing all values greater than cmax. Let $C = ((l, \nu), \{(l_1, v_1, \vec{u}_1), \ldots, (l_m, v_m, \vec{u}_m)\})$ be a joint configuration. To simplify the definition, we write C as a set

$$\{(l, x_1, \nu(x_1), \vec{0}), \ldots, (l, x_n, \nu(x_n), \vec{0}), (l_1, x, v_1, \vec{u}_1), \ldots, (l_m, x, v_m, \vec{u}_m)\}.$$

Partition C into a sequence of subsets $C_0, C_1, \dots, C_\rho, C_\top$, such that $C_\top = \{(k, y, \eta, \vec{\mu}) \in C \mid \eta > \mathsf{cmax}\}$, and if $i, j \neq \top$, then for all $(k, y, \eta, \vec{\mu}) \in C_i$, $(k', y', \eta', \vec{\mu'}) \in C_j$, we have $\mathsf{frac}(\eta) < \mathsf{frac}(\eta')$ if, and only if, $i < j$, and $\mathsf{frac}(\eta) = \mathsf{frac}(\eta')$ if, and only if, $i = j$. Here, $\mathsf{frac}(r)$ denotes the fractional part of a real number r. In this way, $(k, y, \eta, \vec{\mu})$ and $(k', y', \eta', \vec{\mu'})$ are in the same subset C_i if, and only if, η and η' are both smaller than or equal to cmax and have the same fractional part. In addition, we require that $(k, y, \eta, \vec{\mu}) \in C_0$ if, and only if, the fractional part of η is zero, and $C_i \neq \emptyset$ for all $i \in \{1, \dots, \rho\}$. We define the encoding $\mathsf{enc}(C)$ of C to be the finite word $\mathsf{reg}(C_0)\mathsf{reg}(C_1)\dots\mathsf{reg}(C_\rho)\mathsf{reg}(C_\top)$, where $\mathsf{reg}(C_i) = \{(k, y, \mathsf{reg}(\eta), \vec{\mu}) \mid (k, y, \eta, \vec{\mu}) \in C_i\}$ with $\mathsf{reg}(\eta) = \mathsf{int}(\eta)$ if $\eta \leq \mathsf{cmax}$, and $\mathsf{reg}(\eta) = \top$ otherwise ($\mathsf{int}(r)$ denotes the integer part of a real number r).

We define a transition relation $\rightarrow$ on the set of encodings of joint configurations and Σ as follows: $\langle w, a, w' \rangle \in \rightarrow$ if there exists $C \in \mathsf{enc}^{-1}(w)$ and $C' \in \mathsf{enc}^{-1}(w')$ such that $\langle C, a, C' \rangle \in \Rightarrow_{\mathcal{AB}}$. We further define the equivalence relation $\sim$ by $C \sim C'$ if, and only if, $\mathsf{enc}(C) = \mathsf{enc}(C')$.

Example 5.2. *Let $\mathcal{A}$ be a timed automaton with a single clock y, and let $\mathcal{B}$ be a timed one-counter net of dimension 2 and with a single clock x. Assume $\mathsf{cmax} = 2$. Let*

$$C_2 = \langle (l, 1.3), \{(l_2, 0.7, (1,1)), (l_1, 1.0, (1,1)), (l_3, 0.5, (0,1)), (l_1, 2.2, (0,0))\} \rangle$$

be a joint configuration of $\mathcal{A}$ and $\mathcal{B}$. The encoding of C_2 equals

$$\mathsf{enc}(C_2) = \{(l_1, x, 1, (1,1))\}\{(l, y, 1, (0,0))\}\{(l_3, x, 0, (0,1))\}\{(l_2, x, 0, (1,1))\}\{(l_1, x, \top, (0,0))\}$$

The joint configuration

$$C_3 = \langle (l, 1.1), \{(l_2, 0.9, (1,1)), (l_1, 1.0, (1,1)), (l_3, 0.2, (0,1)), (l_1, 9.2, (0,0))\} \rangle$$

has the same encoding, and thus $C_2 \sim C_3$. Note that for $C_2 \sim C_3$ to hold, the configurations must agree on the counter values.

In the next lemma, we prove that $\sim$ is a time-abstract bisimulation over joint configurations. The proof can be done like the proof of Prop. 11 in [34].

Lemma 5.3. *For all joint configurations C_1, C_2, and $a \in \Sigma$, if $C_1 \sim C_2$, then*

- *for all C_1' such that $\langle C_1, a, C_1' \rangle \in \Rightarrow_{\mathcal{AB}}$, there exists C_2' such that $\langle C_2, a, C_2' \rangle \in \Rightarrow_{\mathcal{AB}}$ and $C_1' \sim C_2'$,*
- *for all C_2' such that $\langle C_2, a, C_2' \rangle \in \Rightarrow_{\mathcal{AB}}$, there exists C_1' such that $\langle C_1, a, C_1' \rangle \in \Rightarrow_{\mathcal{AB}}$ and $C_1' \sim C_2'$.*

Next, we define a quasi-order $\sqsubseteq$ on the set of encodings of joint configurations and prove that $\sqsubseteq$ is a well-quasi-order. First, define $\preceq$ on $(\mathcal{L}^{\mathcal{A}} \times \mathcal{X} \times \mathsf{reg} \times \vec{0}) \cup (\mathcal{L}^{\mathcal{B}} \times \{x\} \times \mathsf{reg} \times \mathbb{N}^{\mathsf{n}})$ by $(k, y, \eta, \mu) \preceq (k', y', \eta', \mu')$ if, and only if, $k = k'$, $y = y'$, $\eta = \eta'$, and $\mu \leq^{\mathsf{n}} \mu'$. By Lemma 5.1.1 and the fact that $=$ on the finite set $(\mathcal{L}^{\mathcal{A}} \times \mathcal{X} \times \mathsf{reg}) \cup (\mathcal{L}^{\mathcal{B}} \times \{x\} \times \mathsf{reg})$ and $\leq^{\mathsf{n}}$ on $\mathbb{N}^{\mathsf{n}}$ are well-quasi-orders, $\preceq$ is a well-quasi-order, too. By Lemma 5.1.3, the subset order $\preceq^{\mathcal{P}}$ is a well-quasi-order. Finally, we define $\sqsubseteq$ to be the monotone domination order on $\preceq^{\mathcal{P}}$, and then by Lemma 5.1.2, $\sqsubseteq$ is a well-quasi-order.

Example 5.4. *Let $w_1 = \{(l_1, x, 1, (1,0))\}\{(l, y, 1, (0,0))\}\{(l_2, x, 0, (1,1))\}\rangle$. Then $w_1 \sqsubseteq w_2$, where $w_2 = \mathsf{enc}(C_2)$ from the previous example. Note that the counter values in w_1 may be smaller than the associated counter values in w_2, as it is here the case with the counter values for l_1.*

The next lemma states that $\rightarrow$ is downward-compatible with respect to $\sqsubseteq$.

Lemma 5.5. *If $w_1 \sqsubseteq w_2$ and $\langle w_2, a, w_2'\rangle \in\rightarrow$, then there exists w_1' such that $w_1' \sqsubseteq w_2'$ and $\langle w_1, a, w_1'\rangle \in\rightarrow$.*

Proof. The proof is similar to the proof of Lemma 15 in [34].

Let w_1, w_2, w_2' be such that $w_1 \sqsubseteq w_2$ and $\langle w_2, a, w_2'\rangle \in\rightarrow$. Further let $C_1 = ((l_1, \nu_1), \gamma_1) \in \mathsf{enc}^{-1}(w_1)$, $C_2 = ((l_2, \nu_2), \gamma_2) \in \mathsf{enc}^{-1}(w_2)$, and $C_2' = ((l_2', \nu_2'), \gamma_2') \in \mathsf{enc}^{-1}(w_2')$ be such that $\langle C_2, a, C_2'\rangle \in\Rightarrow_{\mathcal{AB}}$. This implies that there exists $\delta \in \mathbb{R}_{\geq 0}$ with $(l_2', \nu_2') \in \mathsf{Succ}^{\mathcal{A}}((l_2, \nu_2), \delta, a)$ and $\gamma_2' = \mathsf{Succ}^{\mathcal{B}}(\gamma_2, \delta, a)$.

Since $w_1 \sqsubseteq w_2$, we know that there exists a set γ_{minus} of states in $\mathcal{B}$ such that $((l_2, \nu_2), \gamma_{minus}) \sim ((l_1, \nu_1), \gamma_1)$ $(\star)$: γ_{minus} is obtained from choosing a suitable set $\gamma_2'' \subseteq \gamma_2$ such that the encoding of $((l_2, \nu_2), \gamma_2'')$ differs from the encoding of $((l_1, \nu_1), \gamma_1)$ only in that the vectors representing the counter values occurring in γ_2'' may be greater (with respect to $\leq^{\mathfrak{n}}$) than the corresponding vectors in γ_1. γ_{minus} is then the result from subtracting suitable values from the vectors in γ_2'' so that the encoding is equal.

Now let $\gamma_{minus}' = \mathsf{Succ}^{\mathcal{B}}(\gamma_{minus}, \delta, a)$. Then $\langle ((l_2, \nu_2), \gamma_{minus}), a, ((l_2', \nu_2'), \gamma_{minus}')\rangle \in\Rightarrow_{\mathcal{AB}}$ $(\star\star)$. Note that we can add suitable values to the vectors in γ_{minus}' to obtain $\gamma_{minus}' \subseteq \gamma_2'$ $(\star\star\star)$.

From $(\star)$ and $(\star\star)$ it follows by Lemma 5.3 that there exists $C_1' = ((l_1', \nu_1'), \gamma_1')$ such that $\langle C_1, a, C_1'\rangle \in\Rightarrow_{\mathcal{AB}}$ and $C_1' \sim ((l_2', \nu_2'), \gamma_{minus}')$. From the former it follows that $\langle w_1, a, w_1'\rangle \in\rightarrow$, where $w_1' = \mathsf{enc}(C_1')$. From the second and $(\star\star\star)$ it follows that $w_1' \sqsubseteq w_2'$. □

Intuitively, comparing the situation for timed counter nets with the situation for pure timed automata like in Lemma 15 in [34], there may now be transitions that can be executed from configurations encoded by w_2, but that are blocked from configurations encoded by w_1 due to the fact that counter values are too small. This, however, does not cause any trouble, because it results in smaller sets of successor configurations, and thus leading to $w_1' \sqsubseteq w_2'$.

Remark 5.6. *Note that Lemma 5.5 does not hold if the counters in $\mathcal{B}$ can be tested for zero. For instance, consider w_1 and w_2 from the previous example. Assume that in $\mathcal{B}$ there are no edges with source location l_2 and l_3, and the only suitable a-labelled edge with source location l_1 does a zero test on the second counter and leaves all other components unchanged. This yields $\mathsf{Succ}^{\mathcal{B}}(\gamma_2, 0, a) = \emptyset$ and $\mathsf{Succ}^{\mathcal{B}}(\gamma_1, 0, a) = \{(l_1, x, 1, (1, 0))\}$, where γ_i is the set of states of $\mathcal{B}$ in some configuration C_i with $C_i \in \mathsf{enc}^{-1}(w_i)$ for $i = 1, 2$. Assume there is an a-labelled edge in $\mathcal{A}$ with source location l and leaving all components unchanged. Then we have $\langle w_2, a, w_2'\rangle \in\rightarrow$ with $w_2' = \emptyset\{(l, y, 1, (0, 0)\}\emptyset$, $\langle w_1, a, w_1'\rangle \in\rightarrow$ with $w_1' = \{(l_1, x, 1, (1, 0))\}\{(l, y, 1, (0, 0))\}\emptyset$. Note that $w_1' \sqsubseteq w_2'$ does* not *hold. Indeed, the universality problem of (even untimed) one-counter automata is undecidable* [23, 28]. *In Section 6, we prove that this is also the case for timed visibly pushdown one-counter automata. This gives us the precise decidability border for the universality problem.*

Finally, we describe the algorithm to decide $L(\mathcal{A}) \subseteq L(\mathcal{B})$. Like in [34], we solve the language inclusion problem by solving the following reachability problem: in the implicit graph of the encoding of joint configurations and the transition relation $\rightarrow$, is there a path from the encoding of one of the finitely many initial joint configuration to the encoding of a bad joint configuration? Note that we have $L(\mathcal{A}) \subseteq L(\mathcal{B})$ if, and only if, there is no such path. For solving the reachability problem, we compute the unfolding of the graph, starting the computation with the encoding of an initial joint configuration. If for the current node labelled by w, there is along the branch already a node labelled with w' and $w' \sqsubseteq w$, then

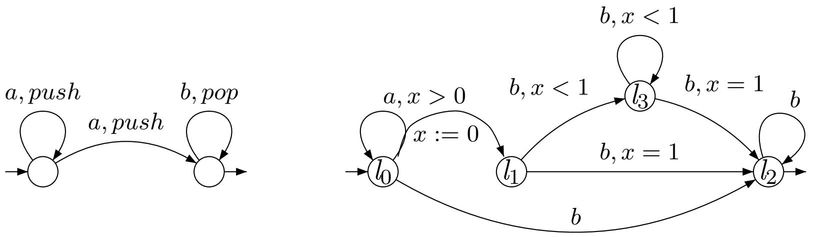

Figure 3: From left to right: A timed one-counter net and an alternating one-clock timed automaton

by Lemma 5.5 we can prune the tree after the current node: Assume that from w we can reach a word w_1 that represents a bad configuration, then by Lemma 5.5 we can reach a word w'_1 from w' such that $w'_1 \sqsubseteq w_1$, and hence, w'_1 is representing a bad configuration, too. By the facts that the unfolding is finitely branching, $\sqsubseteq$ is a well-quasi-order and by König's Lemma, we know that the computation will finally terminate. □

5.2. **On the Expressiveness of Timed Counter Nets.** Theorem 3.2 generalizes a result by Ouaknine and Worrell on the decidability of the language inclusion problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ for $\mathcal{B}$ being a one-clock timed automaton without counters [34]. Clearly, timed counter nets are more expressive than timed automata; for instance, the timed language accepted by the timed one-counter net on the left hand side of Figure 3 cannot be accepted by any timed automaton, because its projection on Σ^+ equals $\{a^n b^m \mid n \geq m, n \geq 1\}$, *i.e.*, a non-regular language.

However, the result in [34] was also generalized to another extension of timed automata called *alternating* one-clock timed automata [35, 32]. An alternating one-clock timed automaton allows for two modes of branching, namely existential branching and universal branching, represented by disjunction and conjunction, respectively. For example, the alternating one-clock timed automaton on the right hand side of Figure 3 has a universal branching transition in l_0 for the input letter a, formally expressed by $\delta(l_0, a) = x > 0 \wedge (l_0 \wedge x.l_1)$; and it has an existential branching transition in l_1 for the input letter b, formally $\delta(l_1, b) = (x < 1 \wedge l_3) \vee (x = 1 \wedge l_2)$ (see [35] for more details). This alternating one-clock timed automaton accepts the timed language consisting of a sequence of a's followed by a sequence of b's such that the time sequence belonging to the a-sequence is strictly monotonic, and every a is followed by some b after exactly one time unit. Note that the projection on Σ^+ thus equals $\{a^n b^m \mid m \geq n, m \geq 1\}$.

We prove that timed one-counter nets with one clock and alternating one-clock timed automata are incomparable in expressive power.

Theorem 5.7. *Timed one-counter one-clock nets and alternating one-clock timed automata are incomparable in expressiveness.*

Proof. On the hand, due to the lack of zero tests, the timed language accepted by the alternating one-clock timed automaton on the right hand side of Figure 3 cannot be accepted by any timed one-counter net. On the other hand, we prove that the timed language $L_{\geq}$ accepted by the timed counter net on the left side of Figure 3 cannot be accepted by any alternating one-clock timed automaton, as we will prove in the following.

We start with some simple facts about deterministic finite automata. Let $\mathcal{B}$ be a deterministic finite automaton over the singleton alphabet $\{b\}$ and with a set of states denoted by Q. For every $n \in \mathbb{N}$, we define a function $f_n^{\mathcal{B}} : Q \to Q$ such that $f_n^{\mathcal{B}}(q) = q'$ means that if $\mathcal{B}$ starts in state q to read the word b^n, then $\mathcal{B}$ ends in q'. Clearly, there exist natural numbers $k, m \geq 1$ such that $f_k^{\mathcal{B}} = f_{k+m}^{\mathcal{B}}$. By determinism of $\mathcal{B}$ we further have $f_{k+i}^{\mathcal{B}} = f_{k+m+i}^{\mathcal{B}}$ for every $i \geq 1$.

Now let $\{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ be a finite set of deterministic finite automata. For every $i \in \{1, \ldots, n\}$, let k_i and m_i be such that $f_{k_i}^{\mathcal{B}_i} = f_{k_i+m_i}^{\mathcal{B}_i}$. Set $k = \max\{k_1, \ldots, k_n\}$, and let m be the least common multiple of $m_1, \ldots, m_n$. One can easily prove that for every $i \in \{1, \ldots, n\}$ we have $f_k^{\mathcal{B}_i} = f_{k+m}^{\mathcal{B}_i}$, and, again by determinism, $f_{k+j}^{\mathcal{B}_i} = f_{k+m+j}^{\mathcal{B}_i}$ for every $j \geq 1$.

Assume by contradiction that $L_{\geq}$ is accepted by an alternating one-clock timed automaton $\mathcal{A}$. Assume $\mathcal{A}$ has n locations. Let B be the set of all deterministic finite automata over $\{b\}$ with at most $2^{2^{2n}}$ states. Note that B is finite up to equivalent behaviour. Now choose $k, m \geq 1$ as explained above and such that $f_{k+j}^{\mathcal{B}} = f_{k+m+j}^{\mathcal{B}}$ for every $\mathcal{B} \in B$ and $j \geq 0$.

Define $\delta = \frac{1}{2k+2m+2}$. Define $w_1 = (a^{k+m}b^{k+1}, \tau)$ and define $w_2 = (a^{k+m}b^{k+m+1}, \tau')$, where $\tau = t_1 t_2 \ldots t_{2k+m+1}$ with $t_i = i \cdot \delta$ for every $i \in \{1, \ldots, 2k+m+1\}$, and, similarly, $\tau' = t'_1 t'_2 \ldots t'_{2k+2m+1}$ with $t'_i = i \cdot \delta$ for every $i \in \{1, \ldots, 2k+2m+1\}$. Note that $w_1 \in L_{\geq}$ and $w_2 \notin L_{\geq}$. Further note that $0 < t_i < 1$ and $0 < t'_i < 1$ for all $i \in \{1, 2k+2m+1\}$.

We prove that $w_1 \in L(\mathcal{A})$ if, and only if, $w_2 \in L(\mathcal{A})$, *i.e.*, $\mathcal{A}$ cannot distinguish between w_1 and w_2.

First assume that $w_1 \in L(\mathcal{A})$. Let γ be the configuration that $\mathcal{A}$ is in after reading a^{k+m}. Clearly, all states (l, x) in γ satisfy $0 \leq x < 1$. Let ρ be the run of $\mathcal{A}$ that starts from γ on the suffix of w_1 that contains the $k+1$ many b's. All clock constraints occurring in transitions of ρ are of the form $x \sim c$ for some $c \in \mathbb{N}$, and by the choice of t_i, the only clock constraints that are relevant for the acceptance of w_1 are those with c equal to 0. The satisfaction of constraints of the form $x \sim 0$ may depend on preceding resets of the clock x; however, even with clock resets occurring in ρ, the clock constraint $x = 0$ cannot be satisfied anywhere in ρ because the time delays between the b's are always greater than 0. In other words, $\mathcal{A}$ behaves on the (untimed) word b^{k+1} like an alternating automaton without a clock, but with an additional flag telling whether there was a reset on the clock or not. This, however, is equivalent to the behaviour of a deterministic finite automaton with $2^{2^{2n}}$ states on the (untimed) word b^{k+1}. But then, by the choice of k and m, we know that starting from γ, $\mathcal{A}$ also accepts b^{k+m+1}, and thus $w_2 \in L(\mathcal{A})$. The proof for the other direction is analogous. □

6. The Universality Problem for Visibly One-Counter Automata

We prove that allowing zero tests in a one-clock timed visibly one-counter net results in the undecidability of the universality problem. The undecidability of the universality problem for the more general class of one-clock visibly pushdown automata was already stated in Theorem 3 in [20]. The proof in [20] is a reduction of the halting problem for two-counter machines. Given a two-counter machine $\mathcal{M}$, one can define a timed language $L(\mathcal{M})$ that consists of all timed words encoding a halting computation of $\mathcal{M}$. Then a timed visibly pushdown automaton $\mathcal{A}$ is defined that accepts the complement of $L(\mathcal{M})$. Altogether, $L(\mathcal{A}) = T\Sigma^+$ if, and only if, $\mathcal{M}$ does not have a halting computation. The definition of $L(\mathcal{M})$ is similar to the definition of $L(\mathcal{C})$ in the proof of Theorem 3.1. Recall that in the definition of $L(\mathcal{C})$ we did not include a condition that requires every symbol to have a matching symbol two time units *before*, and, as we mentioned, this is the reason for $L(\mathcal{C})$ to contain timed words encoding *faulty* computations of $\mathcal{C}$. However, in the definition of $L(\mathcal{M})$ in [20], such a "backward-looking" condition is used. In the proof in [20], it is unfortunately not clear how the one-clock timed visibly pushdown automaton $\mathcal{A}$ can detect violations of this condition[2].

Here, we give a complete proof for the subclass of timed visibly one-counter automata. Like the proof of Theorem 3.1, the proof is a reduction of the control state reachability problem for channel machines. We however remark that one can similarly use a reduction of the halting problem for two-counter machines.

Proof of Theorem 3.4 Let $\mathcal{C} = (S, s_I, M, \Delta)$ be a channel machine, and let $s_F \in S$. Define Σ in the same way as in the proof of Theorem 3.1. For every $n \in \mathbb{N}$, we define a timed language $L_{\mathsf{ef}}(\mathcal{C}, n)$ that consists of all timed words over Σ that encode computations of $\mathcal{C}$ from (s_I, ε) to (s_F, x) for some $x \in M^*$. But in contrast to the proof of Theorem 3.1, $L(\mathcal{C}, n)$ will only contain encodings of *error-free* computations of $\mathcal{C}$.

Formally, $L_{\mathsf{ef}}(\mathcal{C}, n)$ is defined using the same conditions as the ones for $L(\mathcal{C}, n)$ in the proof of Theorem 3.1 plus an additional condition that requires the number of wildcard symbols in the enoding of the last configuration to be equal to n:

(10) Between s_F and $\star$, the wildcard symbol $-$ occurs exactly n times.

Recall that the conditions (1) to (9) guaranteed that the length of consecutive encodings cannot decrease, *i.e.*, the length of every encoding is at least $n + 1$. However, insertion errors may occur, leading to an increase of the length of the encoding and all consecutive encodings. But by the new condition (10), we can exclude the occurrence of such insertion errors. We thus have for $L_{\mathsf{ef}}(\mathcal{C}) = \bigcup_{n \geq 1} L_{\mathsf{ef}}(\mathcal{C}, n)$:

Lemma 6.1. *There exists some error-free computation of* $\mathcal{C}$ *from* (s_I, ε) *to* (s_F, x) *for some* $x \in M^*$*, if, and only if,* $L_{\mathsf{ef}}(\mathcal{C}) \neq \emptyset$.

Next, we define a timed visibly one-counter automaton with a single clock such that $L(\mathcal{A}) = T\Sigma^+ \backslash L_{\mathsf{ef}}(\mathcal{C})$. Hence, by the preceding lemma, $L(\mathcal{A}) \neq T\Sigma^+$ if, and only if, there exists some error-free computation of $\mathcal{C}$ from (s_I, ε) to (s_F, x) for some $x \in M^*$.

$\mathcal{A}$ is the union of $\mathcal{B}$ defined in the proof of Theorem 3.1 and the visibly one-counter automaton shown in Figure 4. The latter accepts timed words violating the new condition (10): The automaton non-deterministically guesses the maximum number n of occurrences of the symbol $+$. When leaving l_1, the value of the counter is $n + 1$. The final location l_4, however, can only be reached while reading $-$ or $\star$ if the value of the counter is zero. This means that the encoding of the last configuration contains at least one symbol more than the encoding of the initial configuration.

7. Conclusion and Open Problems

The main conclusion of this paper is that even for very weak extensions of timed automata with counters it is impossible to automatically verify whether a given specification is satisfied. On the other hand, we may use one-clock timed counter nets as specifications to verify timed automata. The results on the expressive power of timed counter nets in Sect. 5.2 show that this increases so far known possibilities for the verification of timed automata.

[2] More detailed, it is not clear how to construct one-clock timed automata $N_{\neg f_r \leftarrow f_c}$ and $N_{\neg g_r \leftarrow g_c}$ mentioned on p. 10 in [20]. Recall that in the proof for undecidability of the universality problem for timed automata with two or more clocks, it is exactly this backward-looking condition that requires *two* clocks [4].

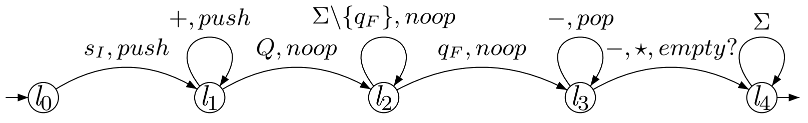

Figure 4: The timed visibly one-counter automaton for recognizing timed words violating the additional "backwards-looking" condition of $L_{\mathsf{ef}}(\mathcal{C})$.

An interesting problem is to figure out a (decidable) extension of LTL that is capable of expressing properties referring to both time and discrete data structures.

We remark that all our results hold for automata defined over *finite* timed words. We cannot expect the decidability of, *e.g.*, the universality problem for one-clock timed counter nets over infinite timed words, as the same problem is already undecidable for the subclass of one-clock timed automata [2].

Acknowledgements. I would like to thank Michael Emmi and Rupak Majumdar for helpful discussions on their work on timed pushdown automata. I further would like to thank James Worrell very much for pointing me to MTL's capability of encoding faulty computations of channel machines.

References

[1] Parosh Aziz Abdulla and Kārlis Čerāns. Simulation is decidable for one-counter nets (extended abstract). In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of *LNCS*, pages 253–268. Springer, 1998.

[2] Parosh Aziz Abdulla, Johann Deneux, Joël Ouaknine, Karin Quaas, and James Worrell. Universality analysis for one-clock timed automata. *Fundam. Inform.*, 89(4):419–450, 2008.

[3] Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theor. Comput. Sci.*, 290(1):241–264, 2003.

[4] Sara Adams, Joël Ouaknine, and James Worrell. Undecidability of universality for timed automata with minimal resources. In Jean-François Raskin and P. S. Thiagarajan, editors, *FORMATS*, volume 4763 of *LNCS*, pages 25–37. Springer, 2007.

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

[6] Rajeev Alur and P. Madhusudan. Decision problems for timed automata: A survey. In Marco Bernardo and Flavio Corradini, editors, *SFM*, volume 3185 of *LNCS*, pages 1–24. Springer, 2004.

[7] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *STOC*, pages 202–211. ACM, 2004.

[8] Ahmed Bouajjani, Rachid Echahed, and Riadh Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems*, volume 999 of *LNCS*, pages 64–85. Springer, 1994.

[9] Florent Bouchy, Alain Finkel, and Arnaud Sangnier. Reachability in timed counter systems. *Electr. Notes Theor. Comput. Sci.*, 239:167–178, 2009.

[10] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. Timed automata with observers under energy constraints. In Karl Henrik Johansson and Wang Yi, editors, *HSCC*, pages 61–70. ACM, 2010.

[11] Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Jirí Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *FORMATS*, volume 5215 of *LNCS*, pages 33–47. Springer, 2008.

[12] Patricia Bouyer, Kim G. Larsen, and Nicolas Markey. Lower-bound-constrained runs in weighted timed automata. *Perform. Eval.*, 73:91–109, 2014.

[13] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, April 1983.

[14] Tomás Brázdil, Petr Jančar, and Antonín Kucera. Reachability games on extended vector addition systems with states. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP (2)*, volume 6199 of *LNCS*, pages 478–489. Springer, 2010.

[15] Rohit Chadha and Mahesh Viswanathan. Decidability Results for Well-Structured Transition Systems with Auxiliary Storage. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *LNCS*, pages 136–150, Springer, 2007.

[16] Zhe Dang. Pushdown timed automata: a binary reachability characterization and safety verification. *Theor. Comput. Sci.*, 302(1-3):93–121, 2003.

[17] Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.

[18] Stéphane Demri, Ranko Lazić, and Arnaud Sangnier. Model checking memoryful linear-time logics over one-counter automata. *Theor. Comput. Sci.*, 411(22-24):2298–2316, 2010.

[19] Leonard Eugene Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Amer. J. Math.*, 35:413–422, 1913.

[20] Michael Emmi and Rupak Majumdar. Decision problems for the verification of real-time software. In J. P. Hespanha and A. Tiwari, editors, *HSCC*, volume 3927 of *LNCS*, pages 200–211. Springer, 2006.

[21] Zoltán Ésik, Uli Fahrenberg, Axel Legay, and Karin Quaas. Kleene algebras and semimodules for energy problems. In Dang Van Hung and Mizuhito Ogawa, editors, *ATVA*, volume 8172 of *LNCS*, pages 102–117. Springer, 2013.

[22] Uli Fahrenberg, Line Juhl, Kim G. Larsen, and Jirí Srba. Energy games in multiweighted automata. In Antonio Cerone and Pekka Pihlajasaari, editors, *ICTAC*, volume 6916 of *LNCS*, pages 95–115. Springer, 2011.

[23] Sheila A. Greibach. An infinite hierarchy of context-free languages. *J. ACM*, 16(1):91–106, 1969.

[24] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. In *LICS*, pages 394–406. IEEE Computer Society, 1992.

[25] Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 2:236–366, 1952.

[26] Piotr Hofman, Slawomir Lasota, Richard Mayr, and Patrick Totzke. Simulation over one-counter nets is PSPACE-complete. In Anil Seth and Nisheeth K. Vishnoi, editors, *FSTTCS*, volume 24 of *LIPIcs*, pages 515–526. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[27] Piotr Hofman and Patrick Totzke. Trace inclusion for one-counter nets revisited. In Joël Ouaknine and Igor Potapov and James Worrell, editors, *RP*, volume 8762 of *LNCS*, pages 151–162. Springer, 2014.

[28] Oscar H. Ibarra. Restricted one-counter machines with undecidable universe problems. *Mathematical Systems Theory*, 13:181–186, 1979.

[29] Petr Jančar, Javier Esparza, and Faron Moller. Petri nets and regular processes. *J. Comput. Syst. Sci.*, 59(3):476–503, 1999.

[30] Petr Jančar, Antonín Kucera, Faron Moller, and Zdenek Sawa. DP lower bounds for equivalence-checking and model-checking of one-counter automata. *Inf. Comput.*, 188(1):1–19, 2004.

[31] Line Juhl, Kim Guldstrand Larsen, and Jean-François Raskin. Optimal bounds for multiweighted and parametrised energy games. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, *Theories of Programming and Formal Methods*, volume 8051 of *LNCS*, pages 244–255. Springer, 2013.

[32] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Log.*, 9(2), 2008.

[33] Information Sciences Institute of the University of Southern California. *Transmission Control Protocoll* (DARPA Internet Program Protocol Specification), 1981. http://www.faqs.org/rfcs/rfc793.html.

[34] Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *LICS*, pages 54–63. IEEE Computer Society, 2004.

[35] Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In *LICS*, pages 188–197. IEEE Computer Society, 2005.

[36] Joël Ouaknine and James Worrell. On metric temporal logic and faulty turing machines. In Luca Aceto and Anna Ingólfsdóttir, editors, *FoSSaCS*, volume 3921 of *LNCS*, pages 217–230. Springer, 2006.

[37] Karin Quaas. On the interval-bound problem for weighted timed automata. In Adrian Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *LATA*, volume 6638 of *LNCS*, pages 452–464. Springer, 2011.

[38] Karin Quaas. Model checking metric temporal logic over automata with one counter. In Adrian Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *LATA*, volume 7810 of *LNCS*, pages 468–479. Springer, 2013.

[39] Karin Quaas. MTL-model checking of one-clock parametric timed automata is undecidable. In Étienne André and Goran Frehse, editors, *SynCoP*, volume 145 of *EPTCS*, pages 5–17, 2014.

[40] Philippe Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. In Inf. Process. Lett., 83(5), pages 251–261, 2002.

[41] Kārlis Čerāns. Decidability of bisimulation equivalences for parallel timer processes. In Gregor von Bochmann and David K. Probst, editors, *CAV*, volume 663 of *LNCS*, pages 302–315. Springer, 1992.