Logical Methods in Computer Science
Volume 00, Number 0, Pages 000–000
S 0000-0000(XX)0000-0

# ON THE DECIDABILITY AND COMPLEXITY OF METRIC TEMPORAL LOGIC OVER FINITE WORDS

JOËL OUAKNINE AND JAMES WORRELL

Oxford University Computing Laboratory, Oxford, UK
*E-mail address*: {joel,jbw}@comlab.ox.ac.uk

Abstract. *Metric Temporal Logic (MTL)* is a prominent specification formalism for real-time systems. In this paper, we show that the satisfiability problem for MTL over finite timed words is decidable, with non-primitive recursive complexity. We also consider the model-checking problem for MTL: whether all words accepted by a given Alur-Dill timed automaton satisfy a given MTL formula. We show that this problem is decidable over finite words. Over infinite words, we show that model checking the safety fragment of MTL—which includes invariance and time-bounded response properties—is also decidable. These results are quite surprising in that they contradict various claims to the contrary that have appeared in the literature.

## 1. Introduction

In the linear-temporal-logic approach to verification, an execution of a system is modelled by a sequence of states or events. This representation abstracts away from the precise times of the observations, retaining only their relative order. Such an approach is inadequate to express specifications of systems whose correct behaviour depends on quantitative timing requirements. To address this deficiency, much work has gone into adapting linear temporal logic to the real-time setting; see, e.g., [6, 7, 9, 10, 23, 26, 31, 34].

Real-time logics feature explicit time references, typically by recording timestamps throughout computations. In this paper, we concentrate exclusively on the *dense-time*, or *real-time*, semantics, in which the timestamps are drawn from the set of real numbers.[1] An important distinction among real-time models is whether one adopts a state-based semantics [7, 20, 31] or an event-based semantics [15, 9, 10, 17, 18, 34]. In the former, an execution of a system is modelled by a function that maps each point in time to the state propositions that are true at that moment. In the latter, one records only a countable sequence of events, corresponding to changes in the state of the system. The distinction between these two semantic models is discussed, among others, in [8, 17]. As we will explain, the main results of this paper crucially depend on our adoption of the event-based model.

*Key words and phrases.* Metric Temporal Logic, Timed Automata, Alternating Automata, Well-quasi-orders.

[1] By contrast, in *discrete-time* settings timestamps are usually integers, which yields more tractable theories that however correspond less closely to physical reality [18, 5].

One of the earliest and most popular proposals for extending temporal logic to the real-time setting is to replace the temporal operators by time-constrained versions—see [8] and the references therein. *Metric Temporal Logic (MTL)*, introduced fifteen years ago by Koymans [23], is a prominent and successful instance of this approach.[2] MTL extends Linear Temporal Logic by constraining the temporal operators by (bounded or unbounded) intervals of the real numbers. For example, the formula $\Diamond_{[3,4]}\varphi$ means that $\varphi$ will hold within 3 to 4 time units from now.

Unfortunately, over the state-based semantics, the satisfiability and model checking problems for MTL are undecidable [15]. This has led some researchers to consider various restrictions on MTL to recover decidability; see, e.g., [6, 7, 18, 34]. Undecidability arises from the fact that MTL formulas can capture the computations of a Turing machine: configurations of the machine can be encoded within a single unit-duration time interval, since the density of time can accommodate arbitrarily large amounts of information. An MTL formula can then specify that the configurations be accurately propagated from one time interval to the next, in such a way that the timed words satisfying the formula correspond precisely to the halting computations of the Turing machine.

It turns out that the key ingredient required for this procedure to go through is *punctuality*: the ability to specify that a particular event is always followed exactly one time unit later by another one: $\Box(p \to \Diamond_{=1} q)$. It has in fact been claimed that, in the state-based and the event-based semantics alike, any logic strong enough to express the above requirement will automatically be undecidable—see [8, 9, 17, 19], among others. While the claim is correct over the state-based semantics, we show in this paper that it is erroneous in the event-based semantics. Indeed, we show that both satisfiability and model checking for MTL over finite timed words are decidable, albeit with non-primitive recursive complexity. Over infinite words, we show that model checking the safety fragment of MTL—which includes invariance and punctual time-bounded response properties—is also decidable.

Upon careful analysis, one sees that the undecidability argument breaks down because, over the event-based semantics, MTL is only able to encode *faulty* Turing machines, namely Turing machines suffering from insertion errors: while the formula $\Box(p \leftrightarrow \Diamond_{=1} q)$ ensures that every $p$ is followed exactly one time unit later by a $q$, there might be some $q$'s that were *not* preceded one time unit earlier by a $p$ (indeed, by any event at all). Intuitively, this problem does not occur over the state-based semantics because the system there is assumed to be under observation at all instants in time, and therefore any insertion error will automatically be detected thanks to the above formula.

MTL is also genuinely undecidable over the event-based semantics if in addition *past* temporal operators are allowed [9, 15]. Indeed, in this setting insertion errors can be detected by going backwards in time, and MTL formulas are therefore able to precisely capture the computations of perfect Turing machines.[3]

The decidability results that we present in this paper are obtained by translating MTL formulas into *timed alternating automata*. These generalise Alur-Dill timed automata, and in particular are closed under complementation. Building on some of our previous work [27], using the theory of *well-structured transition systems*, we show that the language emptiness problem for one-clock timed alternating automata over finite timed words is decidable, which then entails the decidability of MTL satisfiability over finite timed words. We furthermore

[2] As of early 2006, `http://scholar.google.com` lists over three hundred and fifty papers on the subject!

[3] The original undecidability proof in [9] was carried out in a monadic first-order theory of timed words, which subsumes both forward and past temporal operators.

show how to extend these results to the model checking problems discussed earlier. In addition, we show that MTL formulas can capture the computations of insertion channel machines; then, using a result of Schnoebelen about the complexity of reachability for lossy channel machines [32], we give a non-recursive primitive lower bound for the complexity of MTL satisfiability.

1.1. **Related Work.** Existing decidability results for MTL involve placing restrictions on the semantics or the syntax of the logic to circumvent the problem of punctuality. Alur and Henzinger [9] showed that the satisfiability and model checking problems for MTL relative to a discrete-time semantics are EXPSPACE-complete. Alur, Feder, and Henzinger [6, 7] introduced *Metric Interval Temporal Logic (MITL)* as a fragment of MTL in which the temporal operators may only be constrained by *nonsingular* intervals. They showed that the satisfiability and model checking problems for MITL relative to a dense-time semantics are also EXPSPACE-complete. Wilke [34] considered MTL over a dense-time semantics with *bounded variability*, i.e., the semantics is parameterised by a bound $k$ on the number of events per unit time interval. He shows that the satisfiability problem is decidable in this semantics and that MTL with existential quantification over propositions is equally expressive as Alur-Dill timed automata.

A notion of timed alternating automaton very similar to the one considered here has recently and independently been introduced by Lasota and Walukiewicz [24]. They also prove that the finite-word language emptiness problem is decidable for one-clock timed alternating automata, and likewise establish a non-primitive recursive complexity bound for this procedure. However they do not consider any questions related to MTL, or timed logics in general.

Another closely related paper is that of Abdulla and Jonsson [4] on networks of one-clock timed processes. This has a similar flavour to the work presented here in that it uses abstractions based on clock regions and also Higman's Lemma. The problems they study are however very different from the ones considered in this paper.

All the decidability results presented in this paper concern timed alternating automata over finite timed words, including the results that are ostensibly about infinite timed words. In particular, our model checking procedure for the safety fragment of MTL over infinite timed words depends on the fact that any infinite timed word violating a safety property has a finite *bad prefix*, that is, a finite prefix none of whose extensions satisfies the property. Since writing the extended abstract of this paper [28], we have obtained some positive and negative decidability results about the language emptiness problem for timed alternating automata over infinite words. We discuss these results in the Conclusion, Section 9.

## 2. Timed Words and Timed Automata

A *time sequence* $\tau = \tau_1\tau_2\tau_3\ldots$ is a non-empty finite or infinite sequence of time values $\tau_i \in \mathbb{R}_{\geq 0}$ satisfying the following constraints (where $|\tau|$ denotes the length of $\tau$).

- *Initialisation*: $\tau_1 = 0$.
- *Monotonicity*: $\tau_i \leq \tau_{i+1}$ for all $i$, $1 \leq i < |\tau|$.
- *Progress*: if $\tau$ is infinite, then $\{\tau_i : i \in \mathbb{N}\}$ is unbounded.

A *timed word* over finite alphabet $\Sigma$ is a pair $\rho = (\sigma, \tau)$, where $\sigma = \sigma_1\sigma_2\sigma_3\ldots$ is a finite or infinite word over $\Sigma$ and $\tau$ is a time sequence of the same length. We also represent a timed word as a sequence of *timed events* by writing $\rho = (\sigma_1, \tau_1)(\sigma_2, \tau_2)(\sigma_3, \tau_3)\ldots$. Given a timed word $\rho = (\sigma, \tau)$ and $n \leq |\rho|$, let $\rho[1\ldots n]$ denote the prefix $(\sigma_1, \tau_1)\ldots(\sigma_n, \tau_n)$. Finally, write

$T\Sigma^+$ for the set of finite timed words over alphabet $\Sigma$, and $T\Sigma^\omega$ for the set of infinite timed words over $\Sigma$.

The requirement that infinite timed words be progressive is sometimes called *non-Zenoness* or *finite variability*. It is equivalent to the requirement that an infinite number of events not occur in a finite amount of time. Note however that, unlike [34], we place no *a priori* bound on the number of events that can occur in a time interval of unit duration.

Stipulating that the first event of a timed word occur at time 0 is quite natural in the present context, since the semantics of an MTL formula is insensitive to this time value (a similar convention is adopted by Wilke [34]). Also, the convention that timed words be non-empty is in line with the usual model-theoretic practice of excluding models with empty domain. Intuitively one can think of the first position of a timed word as representing an initialisation event.

2.1. **Timed Automata.** Definition 2.1 recalls the standard notion of a *timed automaton* [5]. Elsewhere in this paper we refer to the timed automata defined below as *Alur-Dill automata*. This is to distinguish them from the more general class of *timed alternating automata*, which we introduce in Section 3 and which is our primary focus.

Let $X = \{x_1, \ldots, x_n\}$ be a finite set of *clock variables*. Define the set $\mathcal{C}_X$ of clock constraints over $X$ by the grammar

$$\varphi ::= \top \mid x \bowtie k \mid \varphi_1 \wedge \varphi_2 \,,$$

where $k \in \mathbb{N}$ is a non-negative integer, $x \in X$, and $\bowtie \in \{<, \leq, \geq, >\}$.

**Definition 2.1.** A *timed automaton* is a tuple $\mathcal{A} = (\Sigma, S, s_0, F, X, \Delta)$, where

- $\Sigma$ is a finite alphabet of events
- $S$ is a finite set of locations
- $s_0 \in S$ is an initial location
- $F \subseteq S$ is a set of accepting locations
- $X$ is a finite set of clocks
- $\Delta \subseteq S \times \Sigma \times S \times \mathcal{C}_X \times 2^X$ is a finite set of edges. An edge $(s, a, s', \varphi, R)$ allows $a$-labelled transition from $s$ to $s'$, provided the precondition $\varphi$ on clocks is met. Afterwards, the clocks in $R$ are reset to zero, while all other clocks remain unchanged.

A *clock valuation* of $\mathcal{A}$ is a vector $\mathbf{v} = (v_1, \ldots, v_n)$, where $v_i \in \mathbb{R}_{\geq 0}$ gives the value of clock $x_i$. If $t \in \mathbb{R}_{\geq 0}$, we let $\mathbf{v} + t$ be the clock valuation whose $i$-th component is $v_i + t$. A *state* of $\mathcal{A}$ is a pair $(s, \mathbf{v})$, where $s \in S$ is a location and $\mathbf{v}$ is a clock valuation. Write $Q = S \times (\mathbb{R}_{\geq 0})^n$ for the set of states.

Automaton $\mathcal{A}$ induces a labelled transition system $\mathcal{T}_\mathcal{A} = (Q, \rightsquigarrow, \longrightarrow)$ on the set of states, where $\rightsquigarrow \subseteq Q \times \mathbb{R}_{\geq 0} \times Q$ is called the *flow-step* relation, and $\longrightarrow \subseteq Q \times \Sigma \times Q$ is called the *edge-step* relation. Flow steps model the evolution of time while the automaton remains in a given location, while edge steps corresponds to instantaneous transitions between locations. The flow-step transition relation is deterministic, and is defined by $(s, \mathbf{v}) \overset{t}{\rightsquigarrow} (s, \mathbf{v} + t)$, where $t \in \mathbb{R}_{\geq 0}$. The edge-step relation is defined by $(s, \mathbf{v}) \xrightarrow{a} (s', \mathbf{v}')$ iff there is an edge $(s, a, s', \varphi, R) \in \Delta$ such that $\mathbf{v}$ satisfies $\varphi$, $v'_i = 0$ for all $x_i \in R$ and $v_i = v'_i$ for all $x_i \notin R$.

Let $\rho = (\sigma, \tau)$ be a timed word and write $d_i = \tau_{i+1} - \tau_i$ for the time delay between the $i$-th and $(i+1)$-st events, where $1 \leq i < |\rho|$. Define a *run* of $\mathcal{A}$ on $\rho$ to be an alternating sequence of edge steps and flow steps in $\mathcal{T}_\mathcal{A}$:

$$(s_0, \mathbf{v}_0) \xrightarrow{\sigma_1} (s_1, \mathbf{v}_1) \overset{d_1}{\rightsquigarrow} (s_2, \mathbf{v}_2) \xrightarrow{\sigma_2} (s_3, \mathbf{v}_3) \overset{d_2}{\rightsquigarrow} \cdots \overset{d_{n-1}}{\rightsquigarrow} C_{2n-2} \xrightarrow{\sigma_n} C_{2n-1} \,,$$

where $s_0$ is the initial location and $\mathbf{v}_0$ maps every clock to 0.

A finite run is *accepting* if the last control state in the run is accepting. An infinite run is accepting if infinitely many control states in the run are accepting. We write $L_f(\mathcal{A})$ for the set of finite timed words over which $\mathcal{A}$ has an accepting run, and we write $L_\omega(\mathcal{A})$ for the set of infinite timed words over which $\mathcal{A}$ has an accepting run.

## 3. Timed Alternating Automata

In this section we define timed alternating automata. These arise by extending alternating automata [11, 13, 33] with clock variables, in much the same way that Alur-Dill timed automata extend nondeterministic finite automata. A similar notion has independently been investigated by Lasota and Walukiewicz in a recent paper [24]. It will soon become apparent that timed alternating automata strictly generalise Alur-Dill automata. However we chose to introduce Alur-Dill automata separately in Section 2 since by so doing we can avoid considering timed alternating automata with Büchi acceptance conditions. (This greatly simplifies the definition of a run of an alternating automaton because we can elide the tree structure—see below.)

Timed alternating automata can in general be defined to have any number of clocks. Our goal, however, is to use them to represent metric temporal logic formulas, for which one clock suffices. Accordingly, we shall exclusively focus on one-clock timed alternating automata in this paper.[4] In this section we only consider timed alternating automata over *finite* timed words.

Let $S$ be a finite set of *locations* and let $x$ be a distinguished clock variable. Extending our previous notion of clock constraint, the set of formulas $\Phi(S)$ is generated by the grammar:

$$\varphi ::= \top \mid \bot \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid s \mid x \bowtie k \mid x.\varphi\,,$$

where $k \in \mathbb{N}$, $\bowtie \in \{<, \leq, \geq, >\}$, and $s \in S$.

A term of the form $x \bowtie k$ is called a *clock constraint* and the expression $x.\varphi$ is a binding construct corresponding to the operation of resetting the clock $x$ to 0.

In the definition of a timed alternating automaton, below, the transition function maps each location $s \in S$ and event $a \in \Sigma$ to an expression in $\Phi(S)$. Thus alternating automata allow two modes of branching: existential branching, represented by disjunction, and universal branching, represented by conjunction.

**Definition 3.1.** A timed alternating automaton is a tuple $\mathcal{A} = (\Sigma, S, s_0, F, \delta)$, where

- $\Sigma$ is a finite alphabet
- $S$ is a finite set of locations
- $s_0 \in S$ is the initial location
- $F \subseteq S$ is a set of accepting locations
- $\delta\colon S \times \Sigma \to \Phi(S)$ is the transition function.

The notion of a run of a timed alternating automaton, defined below, is somewhat involved, so we first give an example.

**Example 3.2.** We define an automaton $\mathcal{A}$ over the singleton alphabet $\Sigma = \{a\}$ that accepts all those finite timed words in which no two events are separated by exactly one time unit. This language is known not to be expressible as the language of an Alur-Dill

[4] We note in passing that virtually all decision problems, and in particular language emptiness, are in general undecidable for timed alternating automata that have more than one clock.

timed automaton [21]. The required automaton has set of locations $\{s_0, s_1\}$, with $s_0$ initial, and both $s_0$ and $s_1$ accepting. The transition function is defined by:

$$\delta(s_0, a) = s_0 \wedge x.s_1$$
$$\delta(s_1, a) = s_1 \wedge x \neq 1\,.$$

A run of $\mathcal{A}$ starts in location $s_0$. Every time an $a$-event occurs, the automaton makes a simultaneous transition to both $s_0$ and $s_1$, thus opening up a new thread of computation. The automaton resets a fresh copy of clock $x$ whenever it transitions from location $s_0$ to $s_1$, and ensures that no event can happen when this clock equals 1. Every run of this automaton is accepting, since every location is accepting, but there is no run over any word in which two events are separated by exactly one time unit.

We now proceed to the formal definitions. A *state* of $\mathcal{A}$ is a pair $(s, v)$, where $s \in S$ is a location and $v \in \mathbb{R}_{\geq 0}$ is a *clock valuation*. Write $Q = S \times \mathbb{R}_{\geq 0}$ for the set of all possible states.

A set of states $M \subseteq Q$ and a clock valuation $v \in \mathbb{R}_{\geq 0}$ defines a Boolean valuation on $\Phi(S)$ as follows:

- $M \models_v s$ iff $(s, v) \in M$
- $M \models_v x \bowtie k$ iff $v \bowtie k$
- $M \models_v x.\varphi$ iff $M \models_0 \varphi$.

(The Boolean connectives are handled in the expected way.)

Note that the satisfaction relation is monotone: $N \models_v \varphi$ and $N \subseteq M$ implies $M \models_v \varphi$. We say that $M$ is a *minimal model of* $\varphi \in \Phi(S)$ *with respect to* $v$ if $M \models_v \varphi$ and there is no proper subset $N \subset M$ with $N \models_v \varphi$. Also, if $\varphi \in \Phi(S)$ is a closed formula, i.e., every occurrence of $x$ lies within the scope of a binding operator $x.-$, then the relation $M \models_v \varphi$ is independent of the value of $v$, and we feel free to omit it.

A *configuration* of $\mathcal{A}$ is a finite set of states; the set of configurations is denoted $\wp(Q)$. The *initial configuration* is $\{(s_0, 0)\}$ and a configuration is *accepting* if every location that it contains is accepting. Note in particular that the empty configuration is always accepting. The language accepted by a timed alternating automaton over finite words can be described in terms of a transition system of configurations, defined below.

**Definition 3.3.** Given a timed alternating automaton $\mathcal{A}$, we define the labelled transition system $\mathcal{T}_\mathcal{A} = (\wp(Q), \rightsquigarrow, \longrightarrow)$ over the set of configurations as follows. The $(\mathbb{R}_{\geq 0})$-labelled transition relation $\rightsquigarrow \subseteq \wp(Q) \times \mathbb{R}_{\geq 0} \times \wp(Q)$ captures time evolutions, or *flow steps*, and is defined by

$$C \overset{t}{\rightsquigarrow} C' \text{ if } C' = \{(s, v + t) : (s, v) \in C\}\,.$$

The $\Sigma$-labelled transition relation $\longrightarrow \subseteq \wp(Q) \times \Sigma \times \wp(Q)$ captures instantaneous transitions between locations, or *edge steps*. Let $C = \{(s_i, v_i)\}_{i \in I}$. We include a transition $C \xrightarrow{a} C'$ iff one can choose, for each $i \in I$, a minimal model $M_i$ of $\delta(s_i, a)$ with respect to $v_i$, such that $C' = \bigcup_{i \in I} M_i$.

Let $\rho = (\sigma, \tau)$ be a finite timed word with $|\rho| = n$. Write $d_i = \tau_{i+1} - \tau_i$ for the time delay between the $i$-th and $(i+1)$-st events, $1 \leq i < n$. Define a *run* of $\mathcal{A}$ on $\rho$ to be a finite alternating sequence of edge steps and flow steps in $\mathcal{T}_\mathcal{A}$:

$$C_0 \xrightarrow{\sigma_1} C_1 \overset{d_1}{\rightsquigarrow} C_2 \xrightarrow{\sigma_2} C_3 \overset{d_2}{\rightsquigarrow} \cdots \overset{d_{n-1}}{\rightsquigarrow} C_{2n-2} \xrightarrow{\sigma_n} C_{2n-1}\,,$$

where $C_0$ is the initial configuration. The run is *accepting* if the last configuration $C_{2n-1}$ is accepting, and the timed word $\rho$ is *accepted* by $\mathcal{A}$ if there is some accepting run of $\mathcal{A}$ on $\rho$. We write $L_f(\mathcal{A}) \subseteq T\Sigma^+$ for the language of finite timed words accepted by $\mathcal{A}$.[5]

**Example 3.4.** A time-bounded response property such as 'for every $a$-event there is a $b$-event exactly one time unit later' can be expressed by the following automaton. Let $\mathcal{A}$ have two locations $\{s_0, s_1\}$ with $s_0$ the initial and only accepting location, and transition function $\delta$ given by the following table:

| | $a$ | $b$ |
|---|---|---|
| $s_0$ | $s_0 \wedge x.s_1$ | $s_0$ |
| $s_1$ | $s_1$ | $(x = 1) \vee s_1$ |

Location $s_0$ represents an invariant, and is present in every configuration in any run of $\mathcal{A}$. When an $a$-event occurs, the conjunction in the definition of $\delta(s_0, a)$ results in the creation of a new thread of computation, starting in location $s_1$. Since this location is not accepting, the automaton must eventually leave it. This is only possible if a $b$-event happens exactly one time unit after the new thread was spawned.

3.1. **Duality and Complementation.** The following derivation shows that the class of languages definable by timed alternating automata is closed under complement. Since it is straightforward to show that this class is also closed under union, timed alternating automata are closed under all Boolean operations. The arguments presented here are similar to the untimed case [11, 13].

Given $\varphi \in \Phi(S)$, we define a *dual formula* $\overline{\varphi} \in \Phi(S)$ as follows. The dual of a clock constraint is its negation (e.g., $\overline{x < k} = x \geq k$), whereas each location is self-dual: $\overline{s} = s$ for $s \in S$. For the propositional connectives we have the usual de Morgan dualities: $\overline{\varphi_1 \vee \varphi_2} = \overline{\varphi_1} \wedge \overline{\varphi_2}$ and $\overline{\varphi_1 \wedge \varphi_2} = \overline{\varphi_1} \vee \overline{\varphi_2}$. Finally, clock-resets distribute through the duality operator: $\overline{x.\varphi} = x.\overline{\varphi}$.

Let $\mathcal{A} = (\Sigma, S, s_0, F, \delta)$ be an alternating timed automaton. The complement automaton $\mathcal{A}^c$ is defined by $\mathcal{A}^c = (\Sigma, S, s_0, S \setminus F, \overline{\delta})$, where $\overline{\delta}(s, a) = \overline{\delta(s, a)}$ for each $s \in S$ and $a \in \Sigma$. Thus we take the dual transition function and the complement of the set of accepting locations.

**Proposition 3.5.** *Let $\varphi \in \Phi(S)$, $v \in \mathbb{R}_{\geq 0}$, and let $R \subseteq Q$ be a set of states; then $R \models_v \varphi$ iff $Q \setminus R \not\models_v \overline{\varphi}$.*

*Proof.* The proof is by structural induction on $\varphi$, and is straightforward from the definition of $\overline{\varphi}$. □

**Proposition 3.6.** *$L(\mathcal{A}) \cap L(\mathcal{A}^c) = \emptyset$.*

*Proof.* Suppose that both $\mathcal{A}$ and $\mathcal{A}^c$ have runs on the same timed word $\rho = (\sigma, \tau)$, with $|\rho| = n$. Denote the run of $\mathcal{A}$ by

$$C_0 \xrightarrow{\sigma_1} C_1 \overset{d_1}{\rightsquigarrow} C_2 \xrightarrow{\sigma_2} C_3 \overset{d_2}{\rightsquigarrow} \cdots \overset{d_{n-1}}{\rightsquigarrow} C_{2n-2} \xrightarrow{\sigma_n} C_{2n-1}\,,$$

and denote the run of $\mathcal{A}^c$ by

$$D_0 \xrightarrow{\sigma_1} D_1 \overset{d_1}{\rightsquigarrow} D_2 \xrightarrow{\sigma_2} D_3 \overset{d_2}{\rightsquigarrow} \cdots \overset{d_{n-1}}{\rightsquigarrow} D_{2n-2} \xrightarrow{\sigma_n} D_{2n-1}\,.$$

[5]It is usual to define a run of an alternating automaton as a *tree* of states. However, over finite words the branching structure plays no role in the definition of acceptance, and we simply define a run to be a sequence of configurations, where each configuration represents a given level of the run tree.

We show by induction on $i \leqslant 2n-1$ that $C_i$ and $D_i$ have non-empty intersection. In particular we deduce that $C_{2n-1}$ and $D_{2n-1}$ meet, so the two runs cannot both be accepting since $\mathcal{A}$ and $\mathcal{A}^c$ have disjoint sets of accepting states.

The base case of the induction is just the observation that $C_0 = D_0 = \{(s_0, 0)\}$. For the induction step, suppose that $(s, v) \in C_i \cap D_i$. In case $i = 2j+1$ is odd, that is, the next transition is a flow step, then $(s, v + d_{j+1}) \in C_{i+1} \cap D_{i+1}$. In case $i = 2j$ is even, then $C_{i+1} \models_v \delta(s, \sigma_{j+1})$ and $D_{i+1} \models_v \overline{\delta(s, \sigma_{j+1})}$. But then it follows from Proposition 3.5 that $C_{i+1}$ and $D_{i+1}$ are not disjoint. This completes the induction step. □

**Proposition 3.7.** $L(\mathcal{A}) \cup L(\mathcal{A}^c) = T\Sigma^+$.

*Proof.* We claim that, given a finite timed word $\rho = (\sigma, \tau)$ and a set of states $R \subseteq Q$, either $\mathcal{A}$ has a run on $\rho$ whose last configuration is a subset of $R$, or $\mathcal{A}^c$ has a run on $\rho$ whose last configuration is a subset of $Q \setminus R$. The proposition follows from the claim by taking $R$ to be the set of states in $\mathcal{A}$ whose underlying location is accepting.

We prove the claim by induction on $|\rho|$ as follows. Let $\rho = (\sigma, \tau)$ and $R \subseteq Q$ be given as in the claim, with $|\rho| = n+1$. Also, let $d_n = \tau_{n+1} - \tau_n$ and write

$$\mathit{pred}(R) = \{(s, v) : R \models_{v+d_n} \delta(s, \sigma_{n+1})\}.$$

Observe also that by Proposition 3.5

$$\begin{aligned} Q \setminus \mathit{pred}(R) &= \{(s, v) : R \not\models_{v+d_n} \delta(s, \sigma_{n+1})\} \\ &= \{(s, v) : Q \setminus R \models_{v+d_n} \overline{\delta(s, \sigma_{n+1})}\}. \end{aligned} \tag{3.1}$$

By induction, either $\mathcal{A}$ has a run on $\rho[1 \ldots n]$ whose last configuration $C$ is a subset of $\mathit{pred}(R)$, or $\mathcal{A}^c$ has a run on $\rho[1 \ldots n]$ whose last configuration $D$ is a subset of $Q \setminus \mathit{pred}(R)$. In the former case, it is immediate that we can extend the given run of $\mathcal{A}$ into a run on $\rho$. Indeed, since $C \subseteq \mathit{pred}(R)$, for each $(s, v) \in C$ we can choose a finite subset of $R$ that is a minimal model of $\delta(s, \sigma_{n+1})$ with respect to clock value $v + d_n$. In the latter case, in similar fashion, it follows from (3.1) that $\mathcal{A}^c$ has a run on $\rho$ whose last configuration is a subset of $Q \setminus R$. □

**Corollary 3.8.** *The class of languages definable by timed alternating automata is effectively closed under all Boolean operations.*

## 4. Decidability of Language Emptiness

It is well known that the universality problem for Alur-Dill timed automata is undecidable [5]. Since the class of multi-clock timed alternating automata is closed under complement and includes the class of Alur-Dill automata, the language-emptiness problem for multi-clock timed alternating automata is not decidable either. However we show in this section that if we restrict our attention to alternating automata with single clock, then language emptiness is decidable. The decision procedure that we present is a generalisation of our previous algorithm for deciding universality for one-clock Alur-Dill automata [27].

The language-emptiness problem for a one-clock alternating automaton $\mathcal{A}$ is equivalent to the following reachability question on the derived transition system $\mathcal{T}_{\mathcal{A}}$: 'Is there a path from the initial configuration to an accepting configuration?'. Since $\mathcal{T}_{\mathcal{A}}$ has uncountably many states—indeed each state has uncountably many successors under the flow-step relation—an abstraction is required to explore the state space. Deriving such an abstraction is the subject of the next subsection.

4.1. **The Bisimulation Lemma.** Let $k$ be a non-negative integer. Define an equivalence relation $\sim_k$ on $\mathbb{R}_{\geq 0}$ by $u \sim_k v$ if either $u, v > k$, or $\lceil u \rceil = \lceil v \rceil$ and $\lfloor u \rfloor = \lfloor v \rfloor$. The corresponding set of equivalence classes, or *regions*, is $REG_k = \{\mathsf{r}_0, \mathsf{r}_1, \ldots, \mathsf{r}_{2k+1}\}$, where $\mathsf{r}_{2i} = \{i\}$ for $i \leq k$, $\mathsf{r}_{2i+1} = (i, i+1)$ for $i < k$, and $\mathsf{r}_{2k+1} = (k, \infty)$. Let $reg_k(u)$ denote the equivalence class of $u \geq 0$. In practice we prefer to omit explicit reference to the threshold $k$ in our notation, and infer it from the context. Thus we adopt the convention that whenever $u, v$ are clock values of a timed automaton $\mathcal{A}$, then $u \sim v$ means $u \sim_k v$, where $k$ is the largest constant appearing in the clock constraints of $\mathcal{A}$.

The *fractional part* of a nonnegative real $x \in \mathbb{R}_{\geq 0}$ is $frac(x) = x - \lfloor x \rfloor$. Using this notion we define the relation $\approx$ on $(\mathbb{R}_{\geq 0})^n$—an $n$-dimensional analog of $\sim$, also depending on an invisible threshold $k$—by $\mathbf{u} \approx \mathbf{v}$ iff $u_i \sim v_i$ for each $i \in \{1, \ldots, n\}$ and $frac(u_i) \leq frac(u_j)$ iff $frac(v_i) \leq frac(v_j)$ for all $i, j \in \{1, \ldots, n\}$. Note that $\approx$ has finite index.

The following is a standard result; see, e.g., [5]. Intuitively, it says that $\approx$ is a bisimulation with respect to time evolutions.

**Proposition 4.1.** *Let $\mathbf{u}, \mathbf{v} \in (\mathbb{R}_{\geq 0})^n$ with $\mathbf{u} \approx \mathbf{v}$. Then for all $t \geq 0$ there exists $t' \geq 0$ such that $(\mathbf{u} + t) \approx (\mathbf{v} + t')$.*

**Definition 4.2.** An equivalence relation $R \subseteq \wp(Q) \times \wp(Q)$ is a *time-abstract bisimulation* on $\mathcal{T}_\mathcal{A}$ if $p \; R \; q$ implies

- $(\forall a \in \Sigma)(p \xrightarrow{a} p'$ implies $\exists q'(q \xrightarrow{a} q'$ and $p' \; R \; q'))$
- $(\forall t \in \mathbb{R}_{\geq 0})(p \overset{t}{\rightsquigarrow} p'$ implies $\exists t' \exists q'(q \overset{t'}{\rightsquigarrow} q'$ and $p' \; R \; q'))$.

Note that the definition above does not require that we match flow steps 'on the nose': the durations of the matching transitions can differ. This explains why we call this notion *time-abstract*. Before employing this definition, we take another look at the notion of minimal model underlying the edge-step transition relation.

Any formula $\varphi \in \Phi(S)$ can be written in disjunctive normal form $\varphi \equiv \bigvee_{j \in J} \bigwedge A_j$, where each $A_j$ is a set of terms of the form $s$, $x.s$, and $x \bowtie k$ (which we call *atoms*). The minimal models of $\varphi$ can be read off from the disjunctive normal form as follows. For a set of atoms $A$ and a clock valuation $v \in \mathbb{R}_{\geq 0}$, let $A[v] \subseteq Q$ be the set of states given by $A[v] = \{(s, v) : s \in A\} \cup \{(s, 0) : x.s \in A\}$. Then each minimal model $M$ of $\varphi$ with respect to $v$ has the form $M = A_j[v]$, for some $j \in J$, where $v$ satisfies all the clock constraints in $A_j$.

**Lemma 4.3** (Bisimulation Lemma)**.** *Define the relation $\equiv \; \subseteq \wp(Q) \times \wp(Q)$ by $C \equiv C'$ iff there is a bijection $f : C \to C'$ such that: (i) $f(s, u) = (t, u')$ implies $s = t$ and $u \sim u'$; (ii) if $f(s, u) = (s, u')$ and $f(t, v) = (t, v')$, then $frac(u) \leq frac(v)$ iff $frac(u') \leq frac(v')$. Then $\equiv$ is a time-abstract bisimulation on $\mathcal{T}_\mathcal{A}$.*

*Proof.* Suppose that $C = \{(s_i, u_i)\}_{i \in I}$ and $D = \{(t_i, v_i)\}_{i \in I}$ are configurations of $\mathcal{A}$, and that $f : C \to D$, where $f(s_i, u_i) = (t_i, v_i)$, is a bijection witnessing $C \equiv D$.

*Matching edge transitions.* Suppose $C$ makes an edge transition $C \xrightarrow{a} C'$ for some $a \in \Sigma$. By the above considerations on minimal models we know that $C' = \bigcup_{i \in I} A_i[u_i]$, where, for each $i \in I$, the set of atoms $A_i$ is a clause in the disjunctive normal form expression for $\delta(s_i, a)$. Setting $D' = \bigcup_{i \in I} A_i[v_i]$ we have $D \xrightarrow{a} D'$. (Here we rely on the fact that $u_i \sim v_i$, so $u_i$ and $v_i$ satisfy the same clock constraints.) We also have $C' \equiv D'$ since we can define a bijection $f' : C' \to D'$ by $f'(s, u_i) = (s, v_i)$ and $f'(s, 0) = (s, 0)$.

*Matching flow transitions.* Suppose $C$ makes a flow transition $C \overset{t}{\rightsquigarrow} C'$ for some $t \in \mathbb{R}_{\geq 0}$. Writing $\mathbf{u} = (u_i)_{i \in I}$ and $\mathbf{v} = (v_i)_{i \in I}$, notice that $C \equiv D$ implies that $\mathbf{u} \approx \mathbf{v}$ in the sense of

Proposition 4.1. By that proposition there exists $t'$ with $(\mathbf{u}+t) \approx (\mathbf{v}+t')$. Thus, writing $D' = D + t'$, we have $D \overset{t'}{\rightsquigarrow} D'$ and $C' \equiv D'$. □

4.2. **The transition system $\mathcal{W}_\mathcal{A}$.** As we have already remarked, the transition system $\mathcal{T}_\mathcal{A}$ is, informally speaking, not an effective structure—it has uncountably many states, and each state can make uncountably many transitions. Now, using the Bisimulation Lemma, we isolate a sub-transition-system of $\mathcal{T}_\mathcal{A}$, denoted $\mathcal{W}_\mathcal{A}$, which can be regarded as an *effective skeleton* of $\mathcal{T}_\mathcal{A}$.[6] In particular, $\mathcal{W}_\mathcal{A}$ has only countably many states and is finitely branching; nonetheless, for each configuration of $\mathcal{T}_\mathcal{A}$ there is a bisimilar configuration of $\mathcal{W}_\mathcal{A}$. The key to obtaining a finitely branching transition system here is to realise that, up to bisimulation, only finitely many configurations are reachable via flow steps from a given configuration.

**Definition 4.4.** Let $C \subseteq Q$ be a configuration. If $C$ is non-empty then let $x_{min} = \min\{frac(v) : (s,v) \in C\}$ and $x_{max} = \max\{frac(v) : (s,v) \in C\}$ be respectively the minimum and maximum fractional parts of the clock values appearing in $C$. Now define the *time successor* of $C$ to be the configuration $next(C)$ given by the following clauses.

- If $C = \emptyset$ then $next(C) = C$.
- If $C \neq \emptyset$ and $x_{min} = 0$, then $next(C) = C + (1 - x_{max})/2$.
- If $C \neq \emptyset$ and $x_{min} > 0$ then $next(C) = C + (1 - x_{max})$.

**Example 4.5.** Consider a configuration $C = \{(s, 1.25), (t, 2.5), (s, 0.75)\}$. Then $next(C) = \{(s, 1.5), (t, 2.75), (s, 1)\}$ (in which time has advanced by 0.25 units, and the clock value in $C$ with largest fractional part has moved to a new region). On the other hand, if $C = \{(s, 1), (t, 0.5)\}$, then $next(C) = \{(s, 1.25), (t, 0.75)\}$ (in which the clock value in $C$ with zero fractional part moves to a new region, while all other clock values remain in the same region).

**Definition 4.6.** Define the labelled transition system $\mathcal{W}_\mathcal{A}$ as follows.

- **Alphabet.** The alphabet of $\mathcal{W}_\mathcal{A}$ is $\Sigma \cup \{\varepsilon\}$.
- **States.** The states of $\mathcal{W}_\mathcal{A}$ are those configurations $C \subseteq Q$ in which all clock values are rational (henceforth call such configurations rational).
- **Transitions.** Each state $C$ makes a unique $\varepsilon$-transition to its time successor $next(C)$. For $a \in \Sigma$, we declare that $C \xrightarrow{a} C'$ in $\mathcal{W}_\mathcal{A}$ iff $C \xrightarrow{a} C'$ in $\mathcal{T}_\mathcal{A}$.

Let $(\xrightarrow{\varepsilon})^*$ denote the reflexive transitive closure of the relation $\xrightarrow{\varepsilon}$. The next two properties show that, up to bisimulation, there is no loss in expressiveness in replacing the flow-step transition relation with $(\xrightarrow{\varepsilon})^*$.

**Proposition 4.7.** *Suppose that $C, D \subseteq Q$ are configurations such that $C \equiv D$ and $D$ is rational. Then for any flow step $C \overset{t}{\rightsquigarrow} C'$ there exists a rational configuration $D'$, with $D\ (\xrightarrow{\varepsilon})^*\ D'$ and $C' \equiv D'$.*

*Proof.* Observe that if $C \overset{t}{\rightsquigarrow} C'$, then $C' \equiv next^n(C)$ for some $n \geq 0$. Furthermore, notice that if $C \equiv D$, then $next(C) \equiv next(D)$. It follows that $C' \equiv next^n(D)$ for some $n \geq 0$; but $D\ (\xrightarrow{\varepsilon})^*\ next^n(D)$, so the proposition is established. □

[6] In the extended abstract of this paper $\mathcal{W}_\mathcal{A}$ was described as a *quotient* of $\mathcal{T}_\mathcal{A}$, akin to the clock-region quotient of an Alur-Dill automaton. However in our opinion the technical details are more straightforward under the present approach.

**Proposition 4.8.** *If configuration $C$ is reachable from the initial configuration $C_0$ in $\mathcal{T}_\mathcal{A}$, then there is a rational configuration $C'$, with $C \equiv C'$, such that $C'$ is reachable from $C_0$ in $\mathcal{W}_\mathcal{A}$.*

*Proof.* Given a path in $\mathcal{T}_\mathcal{A}$

$$C_0 \xrightarrow{\sigma_1} C_1 \overset{d_1}{\rightsquigarrow} C_2 \xrightarrow{\sigma_2} C_3 \overset{d_2}{\rightsquigarrow} \cdots \xrightarrow{\sigma_n} C_{2n-1} = C\,,$$

we use Lemma 4.3 and Proposition 4.7 to generate, step by step, a 'matching' path in $\mathcal{W}_\mathcal{A}$

$$C_0 \xrightarrow{\sigma_1} C'_1 \,(\xrightarrow{\varepsilon})^*\, C'_2 \xrightarrow{\sigma_2} C'_3 \,(\xrightarrow{\varepsilon})^* \cdots \xrightarrow{\sigma_n} C'_{2n-1} = C'\,,$$

where $C_i \equiv C'_i$ for $0 \leq i \leq 2n-1$. □

We have now reduced the language-emptiness problem for $\mathcal{A}$ to the following reachability question for $\mathcal{W}_\mathcal{A}$: 'Is there a path from the initial configuration to an accepting configuration?'. In passing from $\mathcal{T}_\mathcal{A}$ to the finitely-branching transition system $\mathcal{W}_\mathcal{A}$ we have achieved a useful rationalisation (no pun intended), however $\mathcal{W}_\mathcal{A}$ still has infinitely many states, albeit only countably many, so it is not obvious that we can decide reachability in this transition system. We bridge this gap by observing the existence of a *well-quasi-order* on the state space of $\mathcal{W}_\mathcal{A}$. This serves in lieu of finiteness to guarantee the termination of a state-exploration algorithm that computes an over-approximation of the set of reachable states. This is described in the next subsection in terms of the theory of *well-structured transition systems* [14].

4.3. **Well-quasi-orders.** Recall that a quasi-order $(W, \preccurlyeq)$ consists of a set $W$ together with a reflexive, transitive relation $\preccurlyeq$. An infinite sequence $w_1, w_2, w_3, \ldots$ in $(W, \preccurlyeq)$ is said to be *saturating* if there exist indices $i < j$ such that $w_i \preccurlyeq w_j$. $(W, \preccurlyeq)$ is a *well-quasi-order (wqo)* if every infinite sequence in $(W, \preccurlyeq)$ is saturating.

Given a quasi-order $(\Lambda, \preccurlyeq)$, the induced *monotone domination order* (or *subword order*) $\sqsubseteq$ on $\Lambda^*$ is defined by $a_1 \ldots a_m \sqsubseteq b_1 \ldots b_n$ if there exists a strictly increasing function $f\colon \{1 \ldots m\} \to \{1, \ldots, n\}$ such that $a_i \preccurlyeq b_{f(i)}$ for all $i \in \{1, \ldots, m\}$.

**Lemma 4.9** (Higman's Lemma [22])**.** *If $(\Lambda, \preccurlyeq)$ is a wqo, then the monotone domination order $\sqsubseteq$ is a wqo on $\Lambda^*$.*

Next we use Higman's Lemma to establish the existence of a well-quasi-order on the state space of the transition system $\mathcal{W}_\mathcal{A}$. The first step is to define a class of *abstract configurations*, which are intended as canonical representatives of $\equiv$-equivalence classes of configurations.

**Definition 4.10.** An *abstract configuration* is a finite word over the alphabet $\Lambda = \wp(S \times REG)$ of nonempty finite subsets of $S \times REG$.

Observe immediately that the alphabet $\Lambda$ under the subset order, being finite, is trivially a wqo. It follows from Lemma 4.9 that the set of abstract configurations is a wqo under the monotone domination order.

Roughly speaking, each (concrete) configuration $C$ of $\mathcal{A}$ gives rise to an abstract configuration as follows. First, $C$ is converted from a set to a list by ordering its elements according to the fractional part of their clock values. Then each clock value is replaced by the region it lies in. Formally, define an *abstraction function* $H : \wp(Q) \to \Lambda^*$, yielding an abstract configuration $H(C)$ for each configuration $C$ as follows. First, lift the function *reg*

to configurations by $reg(C) = \{(s, reg(v)) : (s, v) \in C\}$. Now given a configuration $C$, partition $C$ into a sequence of subsets $C_1, \ldots, C_n$, such that for all $(s, v) \in C_i$ and $(t, v') \in C_j$, $frac(v) \leqslant frac(v')$ iff $i \leqslant j$ (so $(s, v)$ and $(t, v')$ are in the same block $C_i$ iff $v$ and $v'$ have the same fractional part). Then define $H(C) = reg(C_1)reg(C_2) \ldots reg(C_n) \in \Lambda^*$.

**Example 4.11.** Consider the automaton $\mathcal{A}$ from Example 3.2. The maximum clock constant appearing in $\mathcal{A}$ is 1, thus the corresponding regions are $\mathsf{r}_0 = \{0\}$, $\mathsf{r}_1 = (0, 1)$, $\mathsf{r}_2 = \{1\}$ and $\mathsf{r}_3 = (1, \infty)$. Given a concrete configuration $C = \{(s, 1), (t, 0.4), (s, 1.4), (t, 0.8)\}$, the corresponding abstract configuration $H(C)$ is the word $\{(s, \mathsf{r}_2)\}\{(t, \mathsf{r}_1), (s, \mathsf{r}_3)\}\{(t, \mathsf{r}_1)\}$.

Next we show that concrete configurations that map to the same abstract configuration are time-abstract bisimilar.

**Proposition 4.12.** *If $C$ and $C'$ are $\mathcal{A}$-configurations such that $H(C) = H(C')$, then $C$ and $C'$ are bisimilar in $T_{\mathcal{A}}$.*

*Proof.* This follows from the Bisimulation Lemma and the observation that $H(C) = H(C')$ iff $C \equiv C'$. □

Taking stock, we have defined a class of abstract configurations that carries a natural well-quasi-order, and we have shown that abstract configurations are indeed abstract with respect to the notion of time-abstract bisimilarity. Next we show how to exploit these two observations.

4.4. **Well-Structured Transition Systems.** The notion of *well-structured transition system (wsts)* provides a uniform framework for expressing decidability results about a variety of infinite-state systems, including Petri nets, broadcast protocols and lossy channel systems [1, 14]. Definition 4.13 presents a particular variant, called a *downward wsts* in [14].

**Definition 4.13.** A *well-structured transition system* is a triple $\mathcal{W} = (W, \preccurlyeq, \longrightarrow)$, where $(W, \longrightarrow)$ is a finitely-branching (unlabelled) transition system equipped with a wqo $\preccurlyeq$ such that:

- $\preccurlyeq$ is a decidable relation.
- $Succ(w) := \{w' : w \longrightarrow w'\}$ is computable for each $w \in W$.
- $\preccurlyeq$ is *downward compatible*: if $w, v \in W$ with $w \preccurlyeq v$, then for any transition $v \longrightarrow v'$ there exists a matching sequence of transitions $w \,(\longrightarrow)^*\, w'$ with $w' \preccurlyeq v'$.

Note that downwards compatibility allows a single transition of $v$ to be matched by zero or more transitions of $w$.

**Theorem 4.14.** [14, Theorem 5.5] *Let $\mathcal{W} = (W, \preccurlyeq, \longrightarrow)$ be a wsts. Let $V \subseteq W$ be a downward-closed (i.e. $v' \preccurlyeq v$ and $v \in V$ imply $v' \in V$) decidable subset of $W$. Then, given a state $u \in W$, it is decidable whether there is a sequence of transitions starting at $u$ and ending in $V$.*

We now seek to apply Theorem 4.14 to the case at hand.

**Proposition 4.15.** *The transition system $\mathcal{W}_{\mathcal{A}}$ is a wsts (after forgetting the labels on the transitions).*

*Proof.* Define a quasi-order on the set of configurations by $C \preccurlyeq D$ iff $H(C) \sqsubseteq H(D)$, that is, if the word $H(C)$ corresponding to $C$ is a subword of the word $H(D)$ corresponding to $D$. It is straightforward that $\preccurlyeq$ inherits the property of being a well-quasi-order from $\sqsubseteq$.

Moreover $\preccurlyeq$ is a decidable relation on rational configurations, since $H$ is computable on rational configurations and $\sqsubseteq$ is decidable.

It remains to prove that $\preccurlyeq$ is downward compatible. Now suppose $C \preccurlyeq D$ and there is a transition $D \longrightarrow D'$. We show how to produce a matching sequence of transitions for $C$. To this end, it is helpful to first observe that $C \preccurlyeq D$ implies that there is a configuration $E \subseteq D$ with $C \equiv E$. We now consider two cases according to whether the transition $D \longrightarrow D'$ is an edge step or a flow step.

- Suppose that $D \xrightarrow{a} D'$ is an edge step. Since $E \subseteq D$ and the successors of a configuration under edge steps are computed pointwise (cf. Definition 3.3), there is a configuration $E' \subseteq D'$ with $E \xrightarrow{a} E'$. Now $C \equiv E$, so the Bisimulation Lemma yields a transition $C \xrightarrow{a} C'$ with $C' \equiv E'$. But $C' \equiv E'$ and $E' \subseteq D'$ together imply $C' \preccurlyeq D'$.
- Suppose that $D \xrightarrow{\varepsilon} D'$ is a flow step. Then $D' = D + t$ for some $t \geq 0$, and, writing $E' = E + t$, we have $E' \subseteq D'$. By the Bisimulation Lemma, there exists $t' \geq 0$ such that $(C + t') \equiv E'$. As explained in the proof of Proposition 4.7, there is some configuration $C'$ with $C \,(\xrightarrow{\varepsilon})^*\, C'$ and $C' \equiv (C + t')$. Then $C' \equiv (C + t') \equiv E' \subseteq D'$, so $C' \preccurlyeq D'$. This completes the proof.

□

We are now ready to assert one of our main results.

**Theorem 4.16.** *Let $\mathcal{A}$ be a one-clock timed alternating automaton and let $\mathcal{B}$ be an Alur-Dill timed automaton. Then the language-emptiness problem '$L_f(\mathcal{A}) = \emptyset$?' and the language-inclusion problem '$L_f(\mathcal{B}) \subseteq L_f(\mathcal{A})$?' are both decidable.*

*Proof.* Since a configuration of $\mathcal{W}_\mathcal{A}$ is accepting if it only mentions accepting locations of $\mathcal{A}$, the set of accepting configurations of $\mathcal{W}_\mathcal{A}$ is downward-closed with respect to $\preccurlyeq$. By Proposition 4.14 it is decidable whether an accepting configuration of $\mathcal{W}_\mathcal{A}$ is reachable from the initial configuration. In turn this entails, by Proposition 4.7, that it is decidable whether an accepting configuration of $\mathcal{T}_\mathcal{A}$ is reachable from the initial configuration. But this question is equivalent to language emptiness for $\mathcal{A}$. This proves the first assertion of Theorem 4.16. The proof of the second assertion relies on the construction of a wsts representing the execution of $\mathcal{B}$ and $\mathcal{A}$ in parallel. We omit the details since we treat at length essentially the same construction in Section 8, where we consider a closely related language inclusion problem over infinite timed words. □

As noted earlier, these results have recently and independently been obtained by Lasota and Walukiewicz [24], also building on our earlier paper [27].

## 5. Metric Temporal Logic

In this section we define the syntax and semantics of Metric Temporal Logic (MTL). As discussed in the Introduction, there are two different dense-time semantics for MTL: *event-based* and *state-based*, and for our concerns the difference is crucial. Following [15, 9, 10, 17, 18, 34], among others, we adopt an event-based semantics using timed words. A key observation about this semantics is that the temporal connectives quantify over a countable set of positions in a timed word. In contrast, the state-based semantics, adopted in, e.g., [7, 20, 31], associates a state to each point in real time, and the temporal connectives quantify over the whole time domain. In the state-based semantics one can use a formula of the type $\Box(p \leftrightarrow \Diamond_{=1} q)$ to specify a perfect channel, whereas in the event-based semantics the

same formula only specifies a channel with insertion errors (see Section 7). This observation helps understand why MTL is undecidable under the state-based semantics, whereas, at least over finite words, it is decidable in the event-based semantics (Theorem 6.2).

In the event-based semantics the atomic propositions in MTL refer to particular events, and the temporal connectives quantify over future events. This offers a natural idiom for reasoning about real-time behaviours, as we demonstrate in Example 5.3.

**Definition 5.1.** Given an alphabet $\Sigma$ of events, the formulas of MTL are built up from $\Sigma$ by Boolean connectives and time-constrained versions of the **next** operator $\bigcirc$ and the **until** operator $\mathcal{U}$ as follows:

$$\varphi ::= a \mid \top \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc_I \varphi \mid \varphi_1 \,\mathcal{U}_I\, \varphi_2 \,,$$

where $a \in \Sigma$, and $I \subseteq \mathbb{R}_{\geq 0}$ is an open, closed, or half-open interval with endpoints in $\mathbb{N} \cup \{\infty\}$. If $I = [0, \infty)$, then we omit the annotation $I$ in $\bigcirc_I$ and $\mathcal{U}_I$. We also use pseudo-arithmetic expressions to denote intervals. For example, the expression '$\geq 1$' denotes $[1, \infty)$ and '$= 1$' denotes the singleton $\{1\}$.

Additional temporal operators are defined using the usual conventions. We have the *constrained eventually* operator $\Diamond_I \varphi \equiv \top \,\mathcal{U}_I\, \varphi$, and the *constrained always* operator $\Box_I \varphi \equiv \neg\Diamond_I \neg\varphi$. We define a *dual until* operator via the standard duality: $\varphi_1 \,\widetilde{\mathcal{U}}_I\, \varphi_2 \equiv \neg(\neg\varphi_1 \,\mathcal{U}_I\, \neg\varphi_2)$. Finally, we define **end** $\equiv \neg\bigcirc\top$, which is true precisely in the last position of a word.

**Definition 5.2.** Given a (finite or infinite) timed word $\rho = (\sigma, \tau)$ over alphabet $\Sigma$ and an MTL formula $\varphi$, the satisfaction relation $(\rho, i) \models \varphi$ (read $\rho$ satisfies $\varphi$ at position $i$) is defined as follows (note that we include a clause for dual until, even though it is a derived operator, for future reference):

- $(\rho, i) \models a$ iff $\sigma_i = a$
- $(\rho, i) \models \top$
- $(\rho, i) \models \varphi_1 \wedge \varphi_2$ iff $(\rho, i) \models \varphi_1$ and $(\rho, i) \models \varphi_2$
- $(\rho, i) \models \neg\varphi$ iff $(\rho, i) \not\models \varphi$
- $(\rho, i) \models \bigcirc_I \varphi$ iff $i < |\rho|$, $\tau_{i+1} - \tau_i \in I$ and $(\rho, i+1) \models \varphi$
- $(\rho, i) \models \varphi_1 \,\mathcal{U}_I\, \varphi_2$ iff there exists $j \geqslant i$ such that $(\rho, j) \models \varphi_2$, $\tau_j - \tau_i \in I$, and $(\rho, k) \models \varphi_1$ for all $k$ with $i \leqslant k < j$.
- $(\rho, i) \models \varphi_1 \,\widetilde{\mathcal{U}}_I\, \varphi_2$ iff for all $j \geqslant i$ such that $\tau_j - \tau_i \in I$, either $(\rho, j) \models \varphi_2$ or there exists $k$ with $i \leqslant k < j$ and $(\rho, k) \models \varphi_1$.

We say that $\rho$ satisfies $\varphi$, denoted $\rho \models \varphi$, if $(\rho, 1) \models \varphi$. The set of finite models of an MTL formula $\varphi$ is given by $L_f(\varphi) = \{\rho \in T\Sigma^+ : \rho \models \varphi\}$. The set of infinite models of $\varphi$ is given by $L_\omega(\varphi) = \{\rho \in T\Sigma^\omega : \rho \models \varphi\}$.

**Example 5.3.** Consider a set of events $\Sigma = \{req_i, acq_i, rel_i : i = X, Y\}$ denoting the actions of two processes $X$ and $Y$ that request, acquire, and release a lock.

- $\Box(acq_X \rightarrow \Box_{<3} \neg acq_Y)$ says that $Y$ cannot acquire the lock less than 3 seconds after $X$ acquires the lock.
- $\Box(acq_X \rightarrow rel_X \,\widetilde{\mathcal{U}}_{<3}\, \neg acq_Y)$ says that $Y$ cannot acquire the lock less than 3 seconds after $X$ acquires the lock, unless $X$ first releases it.
- $\Box(req_X \rightarrow \Diamond_{<2}(acq_X \wedge \Diamond_{=1} rel_X))$ says that whenever $X$ requests the lock, it acquires the lock within 2 seconds and releases it exactly one second later.

We believe that this example illustrates the convenience of event-based reasoning in the real-time setting, and invite the reader to specify similar properties in the state-based approach.

Every MTL formula can be put into negation normal form, i.e., where negation is only applied to atomic formulas, by using in addition the dual until and **end** operators. This proceeds in the usual manner, the only awkward case being that of the next operator, which is dealt with as follows:

$$\neg \bigcirc_I \varphi \equiv (\bigcirc_I \neg\varphi) \vee (\bigcirc_{I'} \top) \vee (\bigcirc_{I''} \top) \vee \mathbf{end}\,,$$

where $I' \cup I'' = \mathbb{R}_{\geq 0} \setminus I$. In other words, $\bigcirc_I \varphi$ fails just in case $\varphi$ fails in the next position, or the next event occurs outside of $I$, or the end of the word has been reached.

## 6. MTL over Finite Words

In this section we consider the *satisfiability problem* for MTL over finite words: 'Given an MTL formula $\varphi$, is $L_f(\varphi)$ nonempty?'. We also consider the following *model-checking problem*: 'Given an MTL formula $\varphi$ and an Alur-Dill timed automaton $\mathcal{B}$, is it the case that $L_f(\mathcal{B}) \subseteq L_f(\varphi)$?'. In both cases we show decidability by translating the MTL formulas into equivalent one-clock timed alternating automata and invoking Theorem 4.16. We also show that both problems have non-primitive recursive complexity.

6.1. **Decidability.** Given an MTL formula $\varphi$ in negation normal form, we define a one-clock alternating automaton $\mathcal{A}_\varphi$ such that $L_f(\mathcal{A}_\varphi) = L_f(\varphi)$. Since timed alternating automata are closed under union and intersection, and since it is clear how to define $\mathcal{A}_\varphi$ in case $\varphi$ is an atomic formula or the negation of an atomic formula, without loss of generality we assume that the outermost connective in $\varphi$ is $\bigcirc_I$, $\mathcal{U}_I$ or $\widetilde{\mathcal{U}}_I$.

Define the *closure* of $\varphi$, denoted $cl(\varphi)$, to consist of $\varphi$ itself, all subformulas of $\varphi$ whose outermost connective is a temporal modality (including **end**), plus, for each subformula $\bigcirc_I \psi$, an element $(\bigcirc_I \psi)^r$ called the *residual copy* of $\bigcirc_I \psi$.[7] A location is accepting iff it corresponds to a subformula whose outermost connective is $\widetilde{\mathcal{U}}_I$ or **end**.

Recall that a state of $\mathcal{A}_\varphi$ is a pair consisting of a location of $\mathcal{A}_\varphi$, i.e., a subformula of $\varphi$, and a clock value. We define the transition function $\delta$ so that the presence of state $(\psi, 0)$ in a configuration during a run of $\mathcal{A}_\varphi$ ensures that the input word satisfies $\psi$ at the current position. To enforce this requirement, when $\psi$ is encountered the automaton starts a fresh clock and thereafter propagates $\psi$ from configuration to configuration in the run until all

[7]The inclusion of both $\bigcirc_I \psi$ and the residual copy $(\bigcirc_I \psi)^r$ in $cl(\varphi)$ is mainly a convenience to give a more uniform definition of the transition function of $\mathcal{A}_\varphi$, and to help prove the correctness of the translation in Proposition 6.1. In fact, if the formula $\varphi$ itself is not of the form $\bigcirc_I \psi$, then only residual copies of next-subformulas will occur in runs of $\mathcal{A}_\varphi$.

the obligations that it stipulates are discharged. Formally, $\delta$ is defined by:

$$\begin{aligned}
\delta(a,b) &= \begin{cases} \top & \text{if } a = b \\ \bot & \text{if } a \neq b \end{cases} \\
\delta(\psi_1 \vee \psi_2, a) &= \delta(\psi_1, a) \vee \delta(\psi_2, a) \\
\delta(\psi_1 \wedge \psi_2, a) &= \delta(\psi_1, a) \wedge \delta(\psi_2, a) \\
\delta(\psi_1 \,\mathcal{U}_I\, \psi_2, a) &= ((x.\delta(\psi_2, a)) \wedge x \in I) \vee \\
&\quad ((x.\delta(\psi_1, a)) \wedge (\psi_1 \,\mathcal{U}_I\, \psi_2)) \\
\delta(\psi_1 \,\widetilde{\mathcal{U}}_I\, \psi_2, a) &= ((x.\delta(\psi_2, a)) \vee x \notin I) \wedge \\
&\quad ((x.\delta(\psi_1, a)) \vee (\psi_1 \,\widetilde{\mathcal{U}}_I\, \psi_2)) \\
\delta(\bigcirc_I \psi, a) &= x.(\bigcirc_I \psi)^r \\
\delta((\bigcirc_I \psi)^r, a) &= (x \in I) \wedge x.\delta(\psi, a) \\
\delta(\mathbf{end}, a) &= \bot\,.
\end{aligned}$$

**Proposition 6.1.** $L_f(\mathcal{A}_\varphi) = L_f(\varphi)$.

*Proof.* We first show that $L_f(\mathcal{A}_\varphi) \subseteq L_f(\varphi)$. To this end, let $\rho = (\sigma, \tau)$ be a timed word in $L_f(\mathcal{A}_\varphi)$, with $|\rho| = n$. As usual, write $d_i = \tau_{i+1} - \tau_i$ for $1 \leq i < n$. Suppose that $\mathcal{A}_\varphi$ has an accepting run on $\rho$:

$$C_0 \xrightarrow{\sigma_1} C_1 \overset{d_1}{\rightsquigarrow} C_2 \xrightarrow{\sigma_2} C_3 \overset{d_2}{\rightsquigarrow} \cdots \xrightarrow{\sigma_n} C_{2n-1}\,.$$

We claim that for each subformula $\psi$ of $\varphi$ and each $i$ such that $1 \leqslant i \leqslant n$, $(\rho, i) \models \psi$ whenever $C_{2i-1} \models_0 \delta(\psi, \sigma_i)$. We prove this claim by structural induction on $\psi$.

The base case, in which $\psi \equiv a$ or $\psi \equiv \neg a$ for an atomic formula $a$, is immediate. The only non-trivial cases in the induction step are when the outermost connective of $\psi$ is a temporal modality. We consider the cases $\psi \equiv \bigcirc_I \psi_1$ and $\psi \equiv \psi_1 \,\mathcal{U}_I\, \psi_2$; the case for dual until is similar to that for until, and **end** is straightforward.

- $\psi \equiv \bigcirc_I \psi_1$. If $C_{2i-1} \models_0 \delta(\psi, \sigma_i)$ then, since $\delta(\psi, \sigma_i) = x.(\bigcirc_I \psi_1)^r$, we must have $((\bigcirc_I \psi_1)^r, 0) \in C_{2i-1}$. In turn, this entails that $C_{2i+1} \models_0 \delta(\psi_1, \sigma_{i+1})$ and $\tau_{i+1} - \tau_i \in I$. Thus, by the induction hypothesis, we have $(\rho, i{+}1) \models \psi_1$, whence $(\rho, i) \models \bigcirc_I \psi_1$.
- $\psi \equiv \psi_1 \,\mathcal{U}_I\, \psi_2$. Suppose $C_{2i-1} \models_0 \delta(\psi, \sigma_i)$. We consider two possibilities, corresponding to the two disjuncts in the definition of $\delta(\psi, \sigma_i)$. One possibility is that $C_{2i-1} \models_0 \delta(\psi_2, \sigma_i)$ and $0 \in I$. In this case, by the induction hypothesis, we have $(\rho, i) \models \psi_2$, whence $(\rho, i) \models \psi_1 \,\mathcal{U}_I\, \psi_2$. On the other hand, we may have $C_{2i-1} \models_0 \delta(\psi_1, \sigma_i)$ and $(\psi, 0) \in C_{2i-1}$. Then the definition of the transition function $\delta$ ensures that for each successive value of $j \geq i$ we have that $C_{2j-1} \models \delta(\psi_1, \sigma_j)$ and $(\psi, \tau_j - \tau_i) \in C_{2j+1}$ until at some point $C_{2j+1} \models \delta(\psi_2, \sigma_j)$ and $\tau_j - \tau_i \in I$. (Note that the latter must eventually occur since $\psi$ is not an accepting location.) From the induction hypothesis it is clear that this implies that $(\rho, i) \models \psi$. This completes the proof of the claim.

Having proved the claim, we observe that $(\varphi, 0) \in C_0$ (the initial configuration), and so $C_1 \models_0 \delta(\varphi, \sigma_0)$. Thus, applying the claim in the case $i = 0$ and $\psi \equiv \varphi$, we immediately get that $\rho \models \varphi$ whenever $\mathcal{A}_\varphi$ has an accepting run on $\rho$. This completes the proof that $L_f(\mathcal{A}_\varphi) \subseteq L_f(\varphi)$.

It remains to show the converse inclusion. To this end, observe that $\mathcal{A}_{\neg\varphi} = (\mathcal{A}_\varphi)^c$, that is, the automaton representing $\neg\varphi$ is the complement of the automaton representing

$\varphi$. Indeed, the duality that was used to define the transition function of the complement automaton (cf. Section 3.1) corresponds directly to the duality used to define negation in MTL. Now, using the inclusion that we have just proved, we have

$$T\Sigma^+ \setminus L_f(\mathcal{A}_\varphi) = L_f((\mathcal{A}_\varphi)^c) = L_f(\mathcal{A}_{\neg\varphi}) \subseteq L_f(\neg\varphi) = T\Sigma^+ \setminus L_f(\varphi)\,.$$

But this directly gives $L_f(\varphi) \subseteq L_f(\mathcal{A}_\varphi)$, which completes the proof. □

In conjunction with Theorem 4.16, Proposition 6.1 immediately yields:

**Theorem 6.2.** *The satisfiability and the model-checking problems for MTL over finite words are both decidable.*

## 7. Complexity

Using a result of Schnoebelen [32] about channel systems, we prove that the satisfiability problem for MTL has non-primitive recursive complexity.

A *channel machine* consists of a finite-state automaton acting on an unbounded fifo channel, or queue. More precisely, a channel machine is a tuple $\mathcal{C} = (S, M, \Delta)$, where $S$ is a finite set of *control states*, $M$ is a finite set of *messages*, and $\Delta \subseteq S \times \Sigma \times S$ is the transition relation over label set $\Sigma = \{m!, m? : m \in M\}$. A transition labelled $m!$ writes message $m$ to tail of the channel, and a transition labelled $m?$ reads message $m$ from the head of the channel.

We define an operational semantics for channel machines as follows. A *global state* of $\mathcal{C}$ is a pair $\gamma = (s, x)$, where $s \in S$ is the control state and $x \in M^*$ represents the contents of the channel. The rules in $\Delta$ induce a $\Sigma$-labelled transition relation on the set of global states thus: $(s, m!, t) \in \Delta$ yields a transition $(s, x) \xrightarrow{m!} (t, x \cdot m)$ that writes $m \in M$ to the tail of the channel, and $(s, m?, t) \in \Delta$ yields a transition $(s, m \cdot x) \xrightarrow{m?} (t, x)$ that reads $m \in M$ from the head of the channel. If we only allow the transitions indicated above, then we call $\mathcal{C}$ an *error-free* channel machine. A *computation* of such a machine is a finite sequence of transitions between global states

$$(s_0, x_0) \xrightarrow{\alpha_0} (s_1, x_1) \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{n-1}} (s_n, x_n)\,.$$

We also consider channel machines that operate with *insertion errors*. Given $x, y \in M^*$, write $x \sqsubseteq y$ if $x$ can be obtained from $y$ by deleting any number of letters, e.g., $\mathsf{sub} \sqsubseteq \underline{\mathsf{st}}\mathsf{ub}\underline{\mathsf{born}}$, as indicated by the underlining. (This is an instance of the monotone domination order introduced earlier.) Following [32] we model *insertion errors* by extending the transition relation on global states with the following clause: if $(s, x) \xrightarrow{\alpha} (t, y)$, $x' \sqsubseteq x$ and $y \sqsubseteq y'$, then $(s, x') \xrightarrow{\alpha} (t, y')$. Dually, we define *lossy channel machines* by adding a clause: if $(s, x) \xrightarrow{\alpha} (t, y)$, $x \sqsubseteq x'$ and $y' \sqsubseteq y$, then $(s, x') \xrightarrow{\alpha} (t, y')$. The notion of a computation of a channel machine with insertion errors or lossiness errors is defined analogously to the error-free case.

The *control-state reachability problem* asks, given a channel machine $\mathcal{C} = (S, M, \Delta)$ and two distinct control states $s_{init}, s_{fin} \in S$, whether there is a finite computation of $\mathcal{C}$ starting in global state $(s_{init}, \varepsilon)$ and ending in global state $(s_{fin}, x)$ for some $x \in M^*$. This problem was proved to be decidable for lossy channel machines by Abdulla and Jonsson [4]. Later Schnoebelen [32] showed that it has non-primitive recursive complexity. The *dual control-state reachability problem* asks, given a channel machine $\mathcal{C} = (S, M, \Delta)$ and two distinct control states $s_{init}, s_{fin} \in S$, whether there is a finite computation of $\mathcal{C}$ starting in control state $(s_{fin}, x)$ and ending in state $(s_{init}, \varepsilon)$, for some initial channel contents $x \in M^*$.

Note that the difference between the control-state reachability problem and the dual control-state reachability problem depends on whether the initial or final channel is required to be empty. This difference is significant. For instance, the control-state reachability problem is trivial for channel machines with insertion errors. In this case there is a computation from $(s_{init}, \varepsilon)$ to $(s_{fin}, x)$ for some $x \in M^*$ iff there is a path from $s_{init}$ to $s_{fin}$ in the underlying control automaton. Indeed, given such a path we can always construct a matching computation of the channel machine by using insertion errors to ensure that every read-transition along the path is enabled. In contrast, for the dual control-state reachability problem we have the following result.

**Proposition 7.1.** *The dual control-state reachability problem for channel machines with insertion errors has non-primitive recursive complexity.*

*Proof.* Given a channel machine $\mathcal{C} = (S, M, \Delta)$, the *opposite channel machine* is defined by $\mathcal{C}^{\mathrm{op}} = (S, M, \Delta^{\mathrm{op}})$ where

$$\Delta^{\mathrm{op}} = \{(s, m!, t) : (t, m?, s) \in \Delta\} \cup \{(s, m?, t) : (t, m!, s) \in \Delta\}\,.$$

Note that $\mathcal{C}$ has a computation from $(s, x)$ to $(t, y)$ with lossiness errors iff $\mathcal{C}^{\mathrm{op}}$ has a computation from $(t, y^{\mathrm{op}})$ to $(s, x^{\mathrm{op}})$ with insertion errors, where $(-)^{\mathrm{op}} : M^* \to M^*$ reverses the order of a word. Thus the dual control-state reachability problem for $\mathcal{C}$ with insertion errors is equivalent to the control-state reachability problem for $\mathcal{C}^{\mathrm{op}}$ with lossiness errors. But, as we mentioned above, this last problem is known to be decidable with non-primitive recursive complexity. □

**Theorem 7.2.** *The satisfiability and model checking problems for MTL over finite words have non-primitive recursive complexity.*

*Proof.* We give a reduction of the dual control-state reachability problem for channel machines with insertion errors to the satisfiability problem for MTL.

Let $\mathcal{C} = (S, M, \Delta)$ and $s_{init}, s_{fin} \in S$ be an instance of the dual control-state reachability problem. We consider MTL formulas over the set of events $\Sigma = S \cup \{m!, m? : m \in M\}$. We use the formula $\varphi_{CHAN}$ below to capture the behaviour of a channel: every write-event is followed one time unit later by a matching read-event. However, there is no guarantee that every read-event is *preceded* one time unit earlier by a write-event, so the channel may have insertion errors.

$$\varphi_{CHAN} \equiv \textstyle\bigwedge_{m \in M} \Box(m! \to \Diamond_{=1} m?)\,.$$

In order that there be no confusion in terms of matching write-events with their corresponding subsequent read-events, we require that time be strongly monotonic (no two events can occur at the same time). This is captured by the formula $\varphi_{SM}$:

$$\varphi_{SM} \equiv (\bigcirc_{>0} \top)\, \mathcal{U}\, \mathbf{end}\,.$$

We encode the finite control of $\mathcal{C}$ using the formula $\varphi_{CONT}$:

$$\varphi_{CONT} \equiv \bigwedge_{s \in S} (s \to \bigvee_{(s,\mu,t) \in \Delta} (\bigcirc \mu \wedge \bigcirc \bigcirc t))\,.$$

We then use $\varphi_{RUN}$ to assert that a run must start in control state $s_{fin}$ and obey the discrete controller until it terminates in control state $s_{init}$ with empty channel:

$$\varphi_{RUN} \equiv s_{fin} \wedge (\varphi_{CONT}\, \mathcal{U}\, (s_{init} \wedge \mathbf{end}))\,.$$

We combine all these requirements into $\varphi_{REACH}$:

$$\varphi_{REACH} \equiv \varphi_{CHAN} \wedge \varphi_{SM} \wedge \varphi_{RUN} .$$

Suppose we are given a timed word $\rho$ satisfying $\varphi_{REACH}$; then we can construct a computation of $\mathcal{C}$ as follows. First, observe that $\rho$ consists of an alternating sequence of events from $S$ and events from $\{m!, m? : m \in M\}$. This gives the sequence of control states and transitions in the desired computation; it remains to construct the contents of the channel at each control state. Suppose event $s \in S$ occurs at some point along $\rho$ with timestamp $t$. Then the channel contents associated to this occurrence of $s$ is the sequence of read events occurring in $\rho$ in the time interval $(t, t+1)$. Observe how this definition ensures that a message can only be read from the head of the channel, and how each write event adds a message to the tail of the channel. Finally, observe that any timed word satisfying $\varphi_{REACH}$ must have $s_{init}$ as its last event; this ensures that the channel is empty at that point.

Conversely, suppose we are given a computation of $\mathcal{C}$,

$$(s_0, x_0) \xrightarrow{\alpha_0} (s_1, x_1) \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{n-1}} (s_n, x_n)$$

with $s_0 = s_{fin}$, $s_n = s_{init}$ and $x_n = \varepsilon$. We then derive a timed word $\rho = (\sigma, \tau)$ that satisfies $\varphi_{REACH}$. We define $\sigma = s_0\alpha_0 s_1 \alpha_1 \ldots s_n$; this guarantees that $\rho$ satisfies $\varphi_{CONT}$. It remains to choose a sequence of timestamps $\tau$ such that $\varphi_{CHAN} \wedge \varphi_{SM}$ is also satisfied.

Since the given computation of $\mathcal{C}$ ends with the empty channel, every message that is written to the channel is eventually read from the channel. Thus for each write event $m!$ in $\sigma$ there is a 'matching' read event $m?$ later on. We choose the timestamps $\tau$ so that each such matching pair is separated by one time unit. Formally we choose the $\tau_i$ sequentially, starting with $\tau_0 = 0$ and maintaining the following invariant: $\tau_i$ is chosen such that for each matching pair $\sigma_j = m!$ and $\sigma_k = m?$, if $j < k = i$ then $\tau_i - \tau_j = 1$, and if $j < i < k$ then $\tau_i - \tau_j < 1$. It is clearly possible to do this using the density of time.

Thus a channel machine $\mathcal{C} = (S, M, \Delta)$ and pair of control states $s_{init}, s_{fin} \in S$ is a positive instance of the dual reachability problem iff the formula $\varphi_{REACH}$ is satisfiable. This shows that the satisfiability problem for MTL has non-primitive recursive complexity.

Finally, consider a universal Alur-Dill timed automaton, i.e., one that accepts all non-Zeno timed traces. Model checking this automaton against a given MTL formula is equivalent to asking whether the formula is valid, i.e., whether its negation is unsatisfiable. The complexity of model checking MTL is therefore also non-primitive recursive. □

## 8. Infinite Words: Safety MTL

In this section we reuse constructions from Section 4 to prove the decidability of the model-checking problem over infinite words for a subset of MTL, called *Safety MTL*. Safety MTL consists of those MTL formulas in negation normal form that only include instances of the constrained until operator $\mathcal{U}_I$ in which interval $I$ has bounded length. Note that no restrictions are placed on the dual-until operator $\widetilde{\mathcal{U}}_I$.

Safety MTL can express time-bounded response properties, but not arbitrary response formulas. For instance, the formulas $\varphi_1 \equiv \Box(a \rightarrow \Diamond_{=1} b)$ and $\varphi_2 \equiv \Box(a \rightarrow \Diamond_{\leq 5}(b \wedge \Diamond_{=1} c))$ are in Safety MTL, but $\varphi_3 \equiv \Diamond a$ is not. Note in passing that intuitively $\varphi_2$ is much harder to model check than $\varphi_1$. To find a counterexample to $\varphi_1$ one need only guess an $a$-event, and check that there is no $b$-event one time unit later—a task requiring only one clock. On the other hand, to find a counterexample to $\varphi_2$ one must not only guess an $a$-event, but

also check that every $b$-event in the ensuing five time units fails to have a matching $c$-event one time unit later—a task requiring a potentially unbounded number of clocks.

To explain the name Safety MTL, recall from [16] that a language $L \subseteq T\Sigma^\omega$ defines a *safety property relative to the divergence of time* if for every $\rho \notin L$ there exists $n \in \mathbb{N}$ such that no infinite timed word in $T\Sigma^\omega$ extending $\rho[1 \dots n]$ is contained in $L$. In this case we say that $\rho[1 \dots n]$ is a *bad prefix* of $\rho$.

**Proposition 8.1.** *For every Safety MTL formula $\varphi$, $L_\omega(\varphi)$ is a safety property relative to the divergence of time.*

*Proof.* It is straightforward to prove this result by structural induction on $\varphi$. However, we do not give details since we do not use this result in the sequel and since it follows directly, in any case, from Proposition 8.2 and Proposition 8.3. □

To model check a Safety MTL formula $\varphi$ on an Alur-Dill automaton $\mathcal{B}$ we need only check whether any of the bad prefixes of $\varphi$ are prefixes of words accepted by $\mathcal{B}$. We can do this by invoking a variant of the idea used in the proof of Theorem 4.16. To set up this model-checking procedure we first define a translation of $\varphi$ into a one-clock alternating automaton $\mathcal{A}_\varphi^{safe}$ in which every location is accepting.

$\mathcal{A}_\varphi^{safe}$ is a modification of the automaton $\mathcal{A}_\varphi$ from Section 6.1. $\mathcal{A}_\varphi^{safe}$ has the same alphabet, locations and initial location as $\mathcal{A}_\varphi$, but we declare every location of $\mathcal{A}_\varphi^{safe}$ to be accepting. To compensate for this last change, we modify a single clause in the definition of the transition function $\delta$—the clause for $\varphi_1 \,\mathcal{U}_I\, \varphi_2$—as indicated below.

$$\begin{aligned}\delta(\varphi_1 \,\mathcal{U}_I\, \varphi_2, a) = {} & ((x.\delta(\varphi_2, a)) \wedge x \in I) \vee \\ & ((x.\delta(\varphi_1, a)) \wedge (\varphi_1 \,\mathcal{U}_I\, \varphi_2) \wedge (x \leq \sup(I)))\,.\end{aligned}$$

Intuitively, the above definition uses a 'timeout' rather than an acceptance condition to ensure that the second argument of $\mathcal{U}_I$ eventually becomes true. In a non-Zeno run, the automaton cannot get stuck forever in location $\varphi_1 \,\mathcal{U}_I\, \varphi_2$ since the clock constraints in the definition of $\delta(\varphi_1 \,\mathcal{U}_I\, \varphi_2, a)$ only allow transitions when the value of clock $x$ is no greater than $\sup(I)$.

Recall that so far we have only considered alternating automata on finite words. In order to state the correctness of the definition of $\mathcal{A}_\varphi^{safe}$ we consider runs of timed alternating automata on infinite words. Our task is simplified by the fact that we only consider automata in which every location is accepting. (Technically this means that, as with automata over finite words, we can elide the tree structure that is usually associated with runs of alternating automata.) Suppose then that $\mathcal{A}$ is a timed alternating automaton in which every location is accepting. A run of $\mathcal{A}$ on an infinite timed word $\rho = (\sigma, \tau)$ is an infinite alternating sequence of edge steps and flow steps in $\mathcal{T}_\mathcal{A}$:

$$C_0 \xrightarrow{\sigma_1} C_1 \overset{d_1}{\rightsquigarrow} C_2 \xrightarrow{\sigma_2} C_3 \overset{d_2}{\rightsquigarrow} \cdots \overset{d_n}{\rightsquigarrow} C_{2n} \xrightarrow{\sigma_{n+1}} \cdots\,,$$

where $C_0$ is the initial configuration and $d_i = \tau_{i+1} - \tau_i$. We define $L_\omega(\mathcal{A})$ to be the set of $\rho \in T\Sigma^\omega$ over which $\mathcal{A}$ has a run. (Since every state of $\mathcal{A}$ is accepting, there is no need to consider an acceptance condition here.)

**Proposition 8.2.** *$L_\omega(\varphi) = L_\omega(\mathcal{A}_\varphi^{safe})$ for each Safety MTL formula $\varphi$.*

*Proof.* The proof of Proposition 6.1 carries over almost verbatim to the present setting. Referring to the details of that proof, the only change is to observe that it is the 'timeout'

in the definition of $\delta(\varphi_1 \,\mathcal{U}_I\, \varphi_2, a)$, rather than the fact that $\varphi_1 \,\mathcal{U}_I\, \varphi_2$ is non-accepting, that ensures that whenever $(\varphi_1 \,\mathcal{U}_I\, \varphi_2, 0)$ lies in some configuration $C_{2i-1}$ in a run, then there exists $j \geq i$ such that $C_{2j-1} \models \delta(\varphi_2, \sigma_j)$. □

8.1. **The model checking procedure.** Given an Alur-Dill automaton $\mathcal{B}$, and a one-clock timed alternating automaton $\mathcal{A}$ in which every state is accepting, we describe a decision procedure for the model-checking problem '$L_\omega(\mathcal{B}) \subseteq L_\omega(\mathcal{A})$?'. Combining this procedure with Proposition 8.2 gives a method for model checking Safety MTL formulas on Alur-Dill automata.

The following proposition helps enable us to decide whether $L_\omega(\mathcal{B}) \subseteq L_\omega(\mathcal{A})$, while only considering finite runs of $\mathcal{A}$. The idea is that for any word $\rho \in T\Sigma^\omega \setminus L_\omega(\mathcal{A})$, there is a finite bad prefix $\rho[1 \ldots n]$ none of whose (non-Zeno) extensions lies in $L_\omega(\mathcal{A})$.

**Proposition 8.3.** *Let $\mathcal{A}$ be a timed alternating automaton in which every state is accepting. Then $\rho \in T\Sigma^\omega \setminus L_\omega(\mathcal{A})$ iff there exists $n \in \mathbb{N}$ such that $\rho[1 \ldots n] \in L_f(\mathcal{A}^c)$.*

*Proof.* The 'if' direction of the proof is straightforward. Suppose that $\rho[1 \ldots n] \in L_f(\mathcal{A}^c)$.[8] By Proposition 3.6 there can be no run of $\mathcal{A}$ on the finite word $\rho[1 \ldots n]$. (Any such run would be accepting, since every location of $\mathcal{A}$ is accepting.) *A fortiori* there can be no run of $\mathcal{A}$ on $\rho$.

Now we show the 'only if' direction. If $\rho \notin L_\omega(\mathcal{A})$ then $\mathcal{A}$ does not have a run on $\rho$. Moreover we observe that for each $i \geq 1$ there are only finitely many ways to extend a run of $\mathcal{A}$ on the finite prefix $\rho[1 \ldots i]$ to a run on $\rho[1 \ldots (i+1)]$. Thus, by König's lemma, there exists $n \in \mathbb{N}$ such that $\mathcal{A}$ does not have a run on $\rho[1 \ldots n]$. For this choice of $n$ the complement automaton $\mathcal{A}^c$ accepts $\rho[1 \ldots n]$. □

From this point on, the explanation of the model checking procedure closely follows Section 4. In fact, the remainder of this section consists of recapitulations of the definitions and propositions from Section 4, changing what needs to be changed. Briefly, the main difference between Section 4 and the present section is that rather than just considering a wsts generated by a timed alternating automaton, we consider a wsts generated by a timed alternating automaton $\mathcal{A}^c$ and an Alur-Dill automaton $\mathcal{B}$ executing in parallel. We reduce the language emptiness problem '$L_\omega(\mathcal{B}) \cap L_\omega(\mathcal{A}^c) = \emptyset$?' (which is equivalent to '$L_\omega(\mathcal{B}) \subseteq L_\omega(\mathcal{A})$?') to reachability on this wsts.

Suppose that $\mathcal{B}$ is an Alur-Dill timed automaton with $n$ clocks. Recall that a state of $\mathcal{B}$ is a pair $\gamma = (s, \mathbf{v})$, where $s$ is a location of $\mathcal{B}$ and $\mathbf{v} \in (\mathbb{R}_{\geq 0})^n$ is a clock valuation. Given also a one-clock alternating automaton $\mathcal{A}$, define a *$\mathcal{B}$-$\mathcal{A}^c$-configuration* to be a pair $(\gamma, C)$, where $\gamma$ is a state of $\mathcal{B}$ and $C$ is a configuration of $\mathcal{A}^c$. Following the pattern of Definition 4.6 we define a labelled transition system $\mathcal{T}_{\mathcal{B},\mathcal{A}^c}$, representing $\mathcal{B}$ and $\mathcal{A}^c$ executing in parallel.

**Definition 8.4.** The set of states of $\mathcal{T}_{\mathcal{B},\mathcal{A}^c}$ is the set of $\mathcal{B}$-$\mathcal{A}^c$-configurations. Following Definition 4.6 we define an $(\mathbb{R}_{\geq 0})$-labelled flow-step transition relation by $(\gamma, C) \overset{t}{\rightsquigarrow} (\gamma + t, C + t)$ for $t \geq 0$, and a $\Sigma$-labelled edge-step transition relation by $(\gamma, C) \xrightarrow{a} (\gamma', C')$ if $\gamma \xrightarrow{a} \gamma'$ and $C \xrightarrow{a} C'$, where $a \in \Sigma$.

A configuration $(\gamma, C)$ of $\mathcal{T}_{\mathcal{B},\mathcal{A}^c}$ is said to *initial* if $\gamma$ is the initial state of $\mathcal{B}$ and $C$ is the initial configuration of $\mathcal{A}^c$. Recall that $\mathcal{A}^c$ can only accept a word by moving to the

[8] Note that since none of the locations of $\mathcal{A}^c$ is accepting, $\mathcal{A}^c$ can only accept a word by moving to the empty configuration.

empty configuration. Thus a timed word $\rho \in L_\omega(\mathcal{B})$ fails to lie in $L_\omega(\mathcal{A})$ iff there is a computation of $\mathcal{A}^c$ on a finite prefix of $\rho$ that reaches $\emptyset$. Motivated by this observation, we say that a $\mathcal{B}$-$\mathcal{A}^c$-configuration $(\gamma, C)$ is *doomed* if $C = \emptyset$ (i.e., $\mathcal{A}^c$ has reached an accepting configuration) and $\mathcal{B}$ can accept some infinite non-Zeno word starting in state $\gamma$. Then $L_\omega(\mathcal{B}) \not\subseteq L_\omega(\mathcal{A})$ iff there is a doomed configuration $(\gamma, \emptyset)$ that is reachable from the initial configuration of $\mathcal{T}_{\mathcal{B},\mathcal{A}^c}$. Below we sketch how we can use Theorem 4.14 to prove that this reachability problem is decidable.

To set up the application of Theorem 4.14 we reuse constructions from Section 4 to show that $\mathcal{T}_{\mathcal{B},\mathcal{A}^c}$ contains a sub-transition-system $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$ that is a wsts. The first step is to adapt the Bisimulation Lemma, Lemma 4.3, to the present setting. We define an equivalence relation $\equiv$ on $\mathcal{B}$-$\mathcal{A}^c$ configurations that abstracts away from precise clock values, recording only their values to the nearest integer, and the relative order of their fractional parts.

**Definition 8.5.** Given $\mathcal{B}$-$\mathcal{A}^c$ configurations $(\gamma, C)$ and $(\gamma', C')$, define $(\gamma, C) \equiv (\gamma', C')$ if:

- $C \equiv C'$ (in the sense of Lemma 4.3),
- if $\gamma = (s, \mathbf{v})$ and $\gamma' = (s', \mathbf{v}')$, then $s = s'$ and $\mathbf{v} \approx \mathbf{v}'$, and
- if $f : C \to C'$ is the bijection witnessing $C \equiv C'$ and $f(s, u) = (s, u')$ for some $(s, u) \in C$, then $\mathit{frac}(v_i) \bowtie \mathit{frac}(u)$ iff $\mathit{frac}(v'_i) \bowtie \mathit{frac}(u')$ for $1 \le i \le n$ and $\bowtie \in \{<, =, >\}$.

Note that we don't just compare fractional parts among the clock values in $C$, and separately among the clock values in $\gamma$—we also compare between values in $\gamma$ and values in $C$. This is the role of the third clause above.

**Lemma 8.6** (Bisimulation Lemma)**.** *The equivalence relation $\equiv$ in Definition 8.5 is a time-abstract bisimulation on $\mathcal{T}_{\mathcal{B},\mathcal{A}^c}$.*

*Proof.* The significant difference between the proof here and that in Lemma 4.3 concerns matching flow steps. We focus on this and elide the details about matching edge steps, which are straightforward to adapt from Lemma 4.3.

Suppose $(\gamma, C) \equiv (\gamma', C')$ and that there is a flow step $(\gamma, C) \overset{t}{\rightsquigarrow} (\eta, D)$. We must obtain a 'matching' flow step for $(\gamma', C')$.

Write $\gamma = (s, \mathbf{v})$, $\gamma' = (s, \mathbf{v}')$, $C = \{(s_i, u_i)\}_{i \in I}$ and $C' = \{(s_i, u'_i)\}_{i \in I}$ and suppose that the bijection $f : C \to C'$ that witnesses $C \equiv C'$ maps $(s_i, u_i)$ to $(s_i, u'_i)$. Furthermore, write $\mathbf{u} = (u_i)_{i \in I}$ and $\mathbf{u}'_f = (u'_{f(i)})_{i \in I}$, and let $(\mathbf{u} \mid \mathbf{v})$ denote the concatenation of row vectors $\mathbf{u}$ and $\mathbf{v}$. Then we have $(\mathbf{u} \mid \mathbf{v}) \approx (\mathbf{u}'_f \mid \mathbf{v}')$ in the sense of Proposition 4.1. (Note the role played here by the third clause in Definition 8.5 in preserving the ordering of the fractional parts of the respective components of each vector.) By Proposition 4.1, there exists $t'$ such that $((\mathbf{u} \mid \mathbf{v}) + t) \approx ((\mathbf{u}'_f \mid \mathbf{v}') + t')$. Thus, writing $D' = C' + t'$ and $\eta' = \gamma' + t'$ we have $(\gamma', C') \overset{t'}{\rightsquigarrow} (\eta', D')$ and $(\eta, D) \equiv (\eta', D')$. □

Continuing to shadow the development in Section 4, we next define the time successor of a $\mathcal{B}$-$\mathcal{A}^c$-configuration and we define a finitely branching transition system $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$ of *rational configurations*.

**Definition 8.7.** Let $(\gamma, C)$ be a $\mathcal{B}$-$\mathcal{A}^c$-configuration, where $\gamma = (s, \mathbf{v})$, and let $E = \{v_i \mid 1 \le i \le n\} \cup \{v \mid (t, v) \in C\}$ be the set of clock values occurring in $(\gamma, C)$. Write $x_{min} = \min\{\mathit{frac}(v) : v \in E\}$ and $x_{max} = \max\{\mathit{frac}(v) : v \in E\}$ for the respective minimum and maximum fractional parts of the clock values appearing in $E$. Now define the *time*

*successor* of $(\gamma, C)$ to be the configuration $next(\gamma, C) = (\gamma + d, C + d)$, where the time delay $d$ is given by the following clauses.

- If $x_{min} = 0$, then $d = (1 - x_{max})/2$.
- If $x_{min} > 0$, then $d = 1 - x_{max}$.

**Definition 8.8.** Define the labelled transition system $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$ as follows.

- **Alphabet.** The alphabet of $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$ is $\Sigma \cup \{\varepsilon\}$.
- **States.** The states of $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$ are those configurations $(\gamma, C)$ in which all clock values are rational (henceforth call such configurations rational).
- **Transitions.** Each configuration $(\gamma, C)$ makes a unique $\varepsilon$-transition to its time successor $next(\gamma, C)$. For $a \in \Sigma$, we declare that $(\gamma, C) \xrightarrow{a} (\gamma', C')$ in $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$ iff $(\gamma, C) \xrightarrow{a} (\gamma, C')$ in $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$.

**Proposition 8.9.** *If configuration $(\gamma, C)$ is reachable from the initial configuration in $\mathcal{T}_{\mathcal{B},\mathcal{A}^c}$, then there is a rational configuration $(\gamma', C')$, with $(\gamma, C) \equiv (\gamma', C')$, such that $(\gamma', C')$ is reachable from the initial configuration in $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$.*

*Proof.* The proof is almost identical to that of Proposition 4.8, and we omit details. □

To complete the correspondence with Section 4, it remains to show that $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$ is a wsts. As we now explain, this requires a slight variation of the construction used in Proposition 4.15.

Suppose that $\mathcal{A}$ has set of locations $S$ and that $\mathcal{B}$ has set of locations $T$, where $S$ and $T$ are disjoint. Define a finite alphabet

$$\Lambda = \wp(((S \times \{1, \ldots, n\}) \cup T) \times REG_k)\,,$$

where $k$ is the maximum constant mentioned in the clock constraints of $\mathcal{B}$ and $\mathcal{A}$. Following Definition 4.10, an *abstract $\mathcal{B}$-$\mathcal{A}^c$-configuration* is a finite word over $\Lambda$.

We reuse the abstraction function $H$ from Section 4 to map $\mathcal{B}$-$\mathcal{A}^c$-configurations to abstract configurations as follows: map a configuration $((s, \mathbf{v}), C)$ of $\mathcal{T}_{\mathcal{B},\mathcal{A}^c}$ to the word $H(\{((s, 1), v_1), \ldots, ((s, n), v_n)\} \cup C) \in \Lambda^*$. From this word we can reconstruct all clock values in $((s, \mathbf{v}), C)$ up to the nearest integer and also the relative order of the fractional parts of the clocks. As in Proposition 4.12 we use this observation to prove that the kernel of $H$ is a time-abstract bisimulation on $\mathcal{T}_{\mathcal{B},\mathcal{A}^c}$, that is, $H(\gamma, C) = H(\gamma', C')$ implies $(\gamma, C) \equiv (\gamma', C')$.

**Proposition 8.10.** *Define a quasi-order on $\mathcal{B}$-$\mathcal{A}^c$-configurations by $(\gamma, C) \preccurlyeq (\gamma', C')$ iff $H(\gamma, C) \sqsubseteq H(\gamma', C')$, where $\sqsubseteq$ refers to the monotone domination order on $\Lambda^*$. Then $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$ is a wsts when equipped with this quasi-order.*

*Proof.* The proof is almost identical to that of Proposition 4.15. □

**Theorem 8.11.** *Let $\mathcal{B}$ denote an Alur-Dill automaton, and $\mathcal{A}$ a one-clock alternating automaton in which every state is accepting. Then the language inclusion problem '$L_\omega(\mathcal{B}) \subseteq L_\omega(\mathcal{A})$?' is decidable.*

*Proof.* The inclusion $L_\omega(\mathcal{B}) \subseteq L_\omega(\mathcal{A})$ holds iff it is not possible to reach a doomed state from the initial state in $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$. Now the set of doomed states in $\mathcal{W}_{\mathcal{B},\mathcal{A}^c}$ is trivially downward-closed with respect to the monotone domination order (recall that $(\gamma, C)$ is doomed only if $C = \emptyset$). The set of doomed states is also decidable: to decide doomedness of $(\gamma, \emptyset)$ we have to check whether $\mathcal{B}$ can accept a non-Zeno timed word starting from $\gamma$. This last problem

is essentially the language-emptiness problem for Alur-Dill automata over infinite timed words, which is well-known to be decidable—see [5]. Theorem 4.14 now yields a decision procedure for the language inclusion question '$L_\omega(\mathcal{B}) \subseteq L_\omega(\mathcal{A})$?'. □

**Corollary 8.12.** *The model checking problem for Safety MTL over infinite words is decidable: given an Alur-Dill automaton $\mathcal{B}$ and a Safety MTL formula $\varphi$, there is an algorithm to decide whether or not $L_\omega(\mathcal{B}) \subseteq L_\omega(\varphi)$.*

*Proof.* Apply Theorem 8.11 in case $\mathcal{A} = \mathcal{A}_\varphi^{safe}$, using the result of Proposition 8.2 that $L_\omega(\varphi) = L_\omega(\mathcal{A}_\varphi^{safe})$. □

## 9. Conclusion

In this paper, we have shown that Metric Temporal Logic is decidable over finite timed words in its standard dense-time, point-based semantics, with non-primitive recursive complexity. Over infinite words, we have shown that the important safety fragment of Metric Temporal Logic can be model checked, although we do not know the complexity of this problem.

To prove the decidability results above, we introduced the class of timed alternating automata, and showed that the language-emptiness problem for one-clock timed alternating automata over finite words is decidable. In the words of [20], one-clock timed alternating automata constitute a *fully decidable* specification formalism for timed languages in that they are closed under all Boolean operations and language emptiness is decidable. In contrast to Alur-Dill timed automata, one-clock timed alternating automata do not admit finite untimed quotients. In fact, it is straightforward to define a one-clock timed alternating automaton $\mathcal{A}$ such that the untimed language obtained from $L_f(\mathcal{A})$ (by forgetting all timestamps) is the classic non-regular language $\{a^n b^m \mid 0 \leq n \leq m\}$. Reflecting this fact, the termination proof for our language emptiness algorithm used a well-quasi-order derived from Higman's Lemma.

The focus of this paper has exclusively been on MTL over finite words. Recently we have obtained both positive and negative decidability results for MTL over infinite words. In particular, we have shown that the satisfiability problem for Safety MTL is decidable [30], whereas the satisfiability problem for MTL is undecidable [29]. Thus restricting to safety properties is crucial to obtaining decidability.

**Acknowledgements.** We thank Tom Henzinger for clarifying some of the undecidability results for MTL, and for asking about the relationship between single-clock timed automata and real-time temporal logics.

## References

[1] P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. General decidability theorems for infinite-state systems. In *Proceedings of LICS 96*, IEEE Computer Society Press, 1996.

[2] P. A. Abdulla, J. Deneux, J. Ouaknine and J. Worrell. Decidability and complexity results for timed automata via channel systems. In *Proceedings of ICALP 05*, LNCS 3580, 2005.

[3] P. A. Abdulla and B. Jonsson. Undecidable verification problems with unreliable channels. *Information and Computation*, 130:71–90, 1996.

[4] P. A. Abdulla, B. Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*. 290(1):241–264, 2003.

[5] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[6] R. Alur, T. Feder and T. A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of PODC 91*, ACM Press, 1991.

[7] R. Alur, T. Feder and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43:116–146, 1996.
[8] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *Proceedings of Real Time: Theory in Practice*, LNCS 600, 1992.
[9] R. Alur and T. A. Henzinger. Real-time logics: complexity and expressiveness. *Information and Computation*, 104:35–77, 1993.
[10] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41:181–204, 1994.
[11] J. A. Brzozowski and E. Leiss. Finite automata and sequential networks. *Theoretical Computer Science*, 10:19–35, 1980.
[12] G. Cécé, A. Finkel and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124:20–31, 1996.
[13] A. K. Chandra, D. C. Kozen and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.
[14] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.
[15] T. A. Henzinger. The temporal specification and verification of real-time systems. *Ph.D. Thesis*, Technical Report STAN-CS-91-1380, Stanford University, 1991.
[16] T. A. Henzinger. Sooner is safer than later. *Information Processing Letters*, 43:135–141, 1992.
[17] T. A. Henzinger. It's about time: Real-time logics reviewed. In *Proceedings of CONCUR 98*, LNCS 1466, 1998.
[18] T. A. Henzinger, Z. Manna and A. Pnueli. What good are digital clocks? In *Proceedings of ICALP 92*, LNCS 623, 1992.
[19] Y. Hirshfeld and A. M. Rabinovich: Logics for Real Time: Decidability and Complexity. *Fundam. Inform.*, 62(1):1–28, 2004.
[20] T. A. Henzinger, J.-F. Raskin, and P.-Y. Shobbens. The regular real-time languages. In *Proceedings of ICALP 98*, LNCS 1443, 1998.
[21] P. Hermann. Timed automata and recognizability. *Information Processing Letters*, 65:313–318, 1998.
[22] G. Higman. Ordering by divisibility in abstract algebras. *Proc. of the London Mathematical Society*, 2:236–366, 1952.
[23] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time Systems*, 2(4):255–299, 1990.
[24] S. Lasota and I. Walukiewicz. Alternating timed automata. In *Proceedings of FOSSACS 05*, LNCS 3441, 2005.
[25] S. Lasota and I. Walukiewicz. Personal communication, 2005.
[26] J. Ostroff. Temporal logic of real-time systems. Research Studies Press, Taunton, England.
[27] J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *Proceedings of LICS 04*, IEEE Computer Society Press, 2004.
[28] J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *Proceedings of LICS 05*, IEEE Computer Society Press, 2005.
[29] J. Ouaknine and J. Worrell. Metric temporal logic and faulty Turing machines. In *Proceedings of FOSSACS 06*, LNCS 3921, 2006.
[30] J. Ouaknine and J. Worrell. Safety metric temporal logic is fully decidable. In *Proceedings of TACAS 06*, LNCS 3920, 2006.
[31] J.-F. Raskin and P.-Y. Schobbens. State-clock logic: a decidable real-time logic. In *Proceedings of HART 97*, LNCS 1201, 1997.
[32] P. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.
[33] M. Vardi. Alternating automata: Unifying truth and validity checking for temporal logics. In *Proceedings of CADE 97*, LNCS 1249, 1997.
[34] T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. *Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 863, 1994.

Oxford University Computing Laboratory, Oxford, UK
*E-mail address*: {joel,jbw}@comlab.ox.ac.uk