ENS CACHAN

The Cost of Punctuality

Patricia Bouyer, Nicolas Markey,
Joël Ouaknine, and James Worrell

Research report LSV-07-24

Laboratoire Spécification et Vérification

CNRS CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

The Cost of Punctuality

Patricia Bouyer[1,2⋆], Nicolas Markey[1], Joël Ouaknine[2], and James Worrell[2]

[1] LSV – CNRS & ENS Cachan, France
[2] Oxford University, Computing Laboratory, UK

Abstract. In an influential paper titled "The Benefits of Relaxing Punctuality" [AFH96], Alur, Feder, and Henzinger introduced Metric Interval Temporal Logic (MITL) as a fragment of the real-time logic Metric Temporal Logic (MTL) in which exact or punctual timing constraints are banned. Their main result showed that model checking and satisfiability for MITL are both EXPSPACE-Complete.
Until recently, it was widely believed that admitting even the simplest punctual specifications in any linear-time temporal logic would automatically lead to undecidability. Although this was recently disproved, until now no punctual fragment of MTL was known to have even primitive recursive complexity (with certain decidable fragments having provably non-primitive recursive complexity).
In this paper we identify a 'co-flat' subset of MTL that is capable of expressing a large class of punctual specifications and for which model checking (although not satisfiability) has no complexity cost over MITL. Our logic is moreover qualitatively different from MITL in that it can express properties that are not timed-regular. Correspondingly, our decision procedures do not involve translating formulas into finite-state automata, but rather into certain kinds of reversal-bounded Turing machines. Using this translation we show that the model checking problem for our logic is EXPSPACE-Complete.

1 Introduction

In the formal study of real-time systems, it has long been accepted that there is an unavoidable and substantial trade-off between the *expressiveness* of a specification formalism and the *feasability* of the associated verification task. This tension figures most prominently in the case of Metric Temporal Logic (MTL), a timed extension of Linear Temporal Logic (LTL), in which the temporal operators are constrained by time intervals.

MTL was introduced almost two decades ago by Koymans [Koy90], and has since been extensively studied. Unfortunately, the model-checking and satisfiability problems for MTL over dense time are undecidable [AH92,OW06a], an extreme case of infeasibility. Researchers were therefore led to seek relaxations of the framework in a search for tractability. Alur and Henzinger, for example, proved that model checking MTL over discrete time was EXPSPACE-Complete [AH93].

⋆ Supported by a Marie Curie Intra-European Fellowship.

Since untimed LTL model checking is already PSPACE-Complete, their result clearly sat towards the agreeable end of the feasibility spectrum. The price they paid, however, was to renounce the density of time.

Accommodating time density, unfortunately, appeared to be problematic: it was widely held at the time that any formalism in which exact or punctual timing constraints could be expressed would automatically be undecidable. Such constraints correspond to allowing singleton intervals in MTL temporal operators, and enable one to specify, for example, that a particular event is to be followed exactly one time unit later by another one. In their seminal paper titled "The Benefits of Relaxing Punctuality" [AFH96], Alur, Feder, and Henzinger therefore considered a fragment of MTL, called Metric Interval Temporal Logic (MITL), which syntactically bans punctual timing constraints. Their main achievement was to show that the model-checking and satisfiability problems for MITL are EXPSPACE-Complete. The proof they gave, in which MITL formulas are first transformed into timed automata, was quite complicated. Nevertheless, this work was quite influential as it firmly established MITL as the most important fragment of Metric Temporal Logic over dense-time having a feasible model-checking problem. In recent years, new or simplified proofs of the EXPSPACE-Completeness of MITL have appeared in the literature (e.g., [Ras99,HR04,MNP06]).

Recently, it was discovered that punctuality and dense time do not after all necessarily lead to undecidability, although the complexity of the various decidable fragments studied was either non-primitive recursive or non-elementary [OW05,OW06b,OW07]. From a feasibility point of view, such improvements, while significant, remained unsatisfactory.

The aim of the present paper was therefore to investigate more thoroughly the intrinsic cost of allowing punctuality in a dense-time setting. Our main reasults concern two new 'punctual' fragments of Metric Temporal Logic, Bounded-MTL and coFlat-MTL.

Bounded-MTL is derived from MTL by requiring all time-constraining intervals to have finite length. As a result, the truth or falsity of a Bounded-MTL formula on a given timed word is determined by an initial segment of the word whose duration depends solely on the formula. We are then able to show that the model-checking and satisfiability problems for Bounded-MTL over dense time are EXPSPACE-Complete, and PSPACE-Complete if the interval constants are encoded in unary.

Bounded-MTL is therefore a punctual fragment of Metric Temporal Logic having precisely the same complexity as MITL. The two fragments, however, differ in important respects. A first observation is that, at a syntactic level, MITL restricts MTL in banning constraining intervals that are 'too small', whereas Bounded-MTL prohibits intervals that are 'too large'. Semantically, Bounded-MTL thus cannot express invariant properties, required to hold forever, contrary to MITL. In that respect, the expressiveness of Bounded-MTL is quite restricted.

Thankfully, it is possible to incorporate invariance into a substantially larger fragment of MTL. The principal contribution of this paper concerns the logic coFlat-MTL, which subsumes LTL, Bounded-MTL, and is closed under invariance.

Our main result is that model checking this highly expressive punctual fragment of MTL is EXPSPACE-Complete (with constants encoded in binary). In view of the complexity of untimed LTL model checking, such a result can arguably be viewed as optimal. Perhaps surprisingly, satisfiability of coFlat-MTL, on the other hand, turns out to be undecidable.

Our proof of EXPSPACE membership proceeds by translating Flat-MTL formulas into alternating timed automata, and in turn simulating runs of these using special kinds of reversal-bounded Turing machines, for which termination can be shown to be in EXPSPACE. By contrast, MITL formulas are analysed by translation into timed automata, and, unlike Bounded-MTL and coFlat-MTL, can therefore only give rise to timed-regular languages.

MITL and coFlat-MTL have incomparable expressiveness. However, it can be argued that coFlat-MTL comprises virtually all the specifications that one could reasonably be interested to model check in practice. One might therefore view the dense-time logic coFlat-MTL as the first significant fragment of MTL to combine high expressiveness and punctuality together with model-checking feasibility.

The paper is organised as follows: In Section 2, we begin with presenting the model of *channel automata with renaming and occurrence testing* (CAROTs), for which we give the precise complexity of the "cycle-bounded" reachability problem. CAROTs will only be used in Section 5, but they are described and studied in a separate section as we think they are an interesting model that deserves such an emphasis. In Section 3, we define the logic MTL and the fragments we are interested in (especially Flat-MTL and coFlat-MTL). We compare their relative expressiveness and give an overview of the results proven in this paper. We then present in Section 4 a (rather classical) transformation from MTL to alternating timed automata, and give specific properties of the resulting automaton when the original formula is in Flat-MTL. In Section 5, we propose a translation from alternating timed automata to CAROTs, which is roughly the timed counterpart of the transformation of an untimed alternating automaton into a Büchi automaton. In Section 6, we summarize all the constructions and obtain an EXPSPACE algorithm for model checking of coFlat-MTL and satisfiability checking of Flat-MTL. Finally, in Section 7, we prove that those algorithms match the exact complexity of those problems.

2 Channel Automata

Before introducing our real-time modelling framework, we introduce a class of discrete machines that ultimately underly our model-checking algorithm for coFlat-MTL.

A *channel automaton* is a finite-state automaton equipped with a single unbounded fifo channel (or queue). The transitions of the automaton either write messages to the tail of the channel or read messages from the head of the channel. This model is easily seen to be Turing-powerful [BZ83]. In this paper we consider a class of channel automata with two extra primitives: *global renaming* and *occurrence testing*. The former allows a transition to simultaneously rename all

the letters on the channel according to some renaming relation, including the possibility of deleting letters. The latter allows a transition to be guarded by the predicate that some letter not appear on the channel.

Given an alphabet Σ, let Σ_ε denote $\Sigma \cup \{\varepsilon\}$, where ε represents the empty word.

Definition 1. *A* Channel Automaton with Renaming and Occurrence Testing *(CAROT) is a tuple $\mathcal{C} = (S, s_0, \Sigma, \Delta, F)$, where S is a finite set of control states, $s_0 \in S$ is the initial control state, $F \subseteq S$ is a set of final control states, Σ is a finite channel alphabet and $\Delta \subseteq S \times Op \times S$ is the set of transition rules, with*

$$Op = \{\sigma!, \sigma? \mid \sigma \in \Sigma\} \cup \{zero(\sigma) \mid \sigma \in \Sigma\} \cup \{R \mid R \subseteq \Sigma \times \Sigma_\varepsilon\}$$

the set of operations. Given a rule $\tau \in \Delta$, we denote the corresponding operation $op(\tau)$. Intuitively, $zero(\sigma) \in Op$ guards against the occurrence of σ in the channel, and $R \in Op$ is interpreted as a global renaming (where renaming to ε corresponds to deletion).

A *global state* of $\mathcal{C}$ is a pair $\gamma = (s, x)$, where $s \in S$ is the control state and $x \in \Sigma^*$ is the channel contents. The rules in Δ induce a transition relation on the set of global states according to the following table, where, given $x = x_1 \dots x_n \in \Sigma^*$ and $R \subseteq \Sigma \times \Sigma_\varepsilon$, $R(x) \stackrel{\text{def}}{=} \{y_1 \cdots y_n \in \Sigma^* : x_i \; R \; y_i\}$.

Rule	Transition
$(s, \sigma!, t)$	$(s, x) \to (t, x \cdot \sigma)$
$(s, \sigma?, t)$	$(s, \sigma \cdot x) \to (t, x)$
$(s, zero(\sigma), t)$	$(s, x) \to (t, x)$, if $\sigma \notin x$
(s, R, t)	$(s, x) \to (t, y)$, if $y \in R(x)$

Assume that Σ always contains a special symbol $\triangleright$, called the *end-of-channel marker*. A computation of $\mathcal{C}$ is a (finite or infinite) sequence of transitions $\gamma_0 \to \gamma_1 \to \gamma_2 \to \cdots$ with $\gamma_0 = (s_0, \triangleright)$. A finite computation is *accepting* if it ends in an accepting state γ_n.

To aid our analysis of computations, we make the following (harmless) assumption about $\mathcal{C}$: we suppose that given two *consecutive* rules (s, op_1, t) and (t, op_2, u), $op_1 = \triangleright?$ iff $op_2 = \triangleright!$. Roughly speaking, there is always a unique copy of $\triangleright$ on the channel. This restriction allows us to use the notion of the end-of-channel marker to measure the number of *cycles* of the channel during a computation. Intuitively a segment of the computation during which $\triangleright$ moves from the tail of the channel to the head of the channel involves a complete cycle of the channel. Formally we define $cycles(\varrho)$ to be the number of transitions in ϱ with operand $\triangleright!$. This measure is similar to the notion of head reversals for Turing machines.[3]

[3] Formally, it can be shown that an N-cycle-bounded CAROT can simulate an N-reversal-bounded single-tape Turing machine, and *vice-versa*.

Definition 2. *The* cycle-bounded reachability problem *for CAROTs is as follows:*
***Instance**: A CAROT $\mathcal{C}$ and a cycle bound N.*
***Question**: Does $\mathcal{C}$ have an accepting computation ϱ with $\mathit{cycles}(\varrho) \leqslant N$?*

In the channel automaton $\mathcal{C}$ below, let R be the relation that nondeterministically renames b to either b or c.

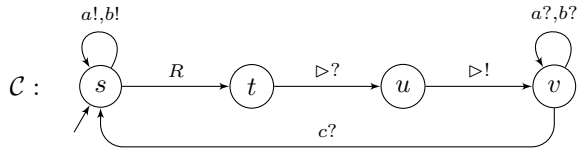


Figure 1 represents a computation of $\mathcal{C}$ in tabular form. Each row of the table represents a cycle of the channel, and, reading left-to-right, it records the sequence of transitions during that cycle. The most important property of the table is that the spacing is arranged so that an operation that reads a message is placed directly below the operation that originally wrote the message, necessarily in the previous cycle of the channel.[4] In Figure 1 matching pairs of reads and writes are indicated by rectangular boxes. Because of global renaming the corresponding read and write events may not refer to the same element of Σ. For instance, in Figure 1, a write-event $b!$ is sometimes aligned with a read-event $c?$.

s	$b!$	s	$b!$	s	R	t	$\triangleright?$	u				$\triangleright!$	v									
	$b?$	v	$c?$	s	$a!$	s	$b!$	s	R	t		$\triangleright?$	u	$\triangleright!$	v							
					$a?$	v	$c?$	s	$b!$	s	R	t		$\triangleright?$	u	$\triangleright!$	v					
									$c?$	s	R	u				$\triangleright?$	u	$\triangleright!$	v			

Fig. 1. Computation table

s	$b!$	s	$b!$	s	R	t	$\triangleright?$	u	$-$	u	$\triangleright!$	v	$-$	v	$-$	v	$-$	v				
v	$b?$	v	$c?$	s	$a!$	s	$b!$	s	R	t	$\triangleright?$	u	$\triangleright!$	v	$-$	v	$-$	v				
v	$-$	v	$-$	v	$a?$	v	$c?$	s	$b!$	s	R	t	$\triangleright?$	u	$\triangleright!$	v	$-$	v				
v	$-$	v	$-$	v	$-$	v	$-$	v	$c?$	s	R	u	$-$	u	$\triangleright?$	u	$\triangleright!$	v				

Fig. 2. Computation table with sliding window

The length of the computation table (*i.e.*, the number of columns) is at least the maximum length of the channel during the corresponding computation. It is easy to see that this can be exponential in the value of the cycle bound. (Consider a machine that repeatedly reads one copy of σ and writes two copies of σ.) However we describe a procedure to guess the existence of a computation table

[4] To accommodate global deletion in such a table (i.e., renaming to ε), we postulate a self-loop $(s, \varepsilon?, s)$ for each control state s of $\mathcal{C}$.

using only polynomial space in the value of the cycle bound. The first step is to fill in the blank spaces in the table by repeating the immediately preceding control state; for example, starting from Figure 1 we obtain the table in Figure 2.

A nondeterministic procedure for guessing and verifying such a table involves storing only part of the table in memory at any one time. Imagine a sliding window of dimension $3 \times h$, where h is the table height (*i.e.*, the number of rows). The window represents the part of the table in view at any time; it starts at the left end and is moved one place to the right with each phase of the procedure. Given a particular view, a phase of the procedure checks that the transitions therein are consistent with the control structure of $\mathcal{C}$. For instance, in Figure 2, while viewing the leftmost window it is checked that $(s, b!, s), (v, b?, v) \in \Delta$. Somewhat more subtly the corresponding read and write events in the current view must be consistent with the zero testing and renaming in the rest of the table. For instance, in Figure 2, again in the left-most window, the justification of the vertically aligned $b!$ and $b?$ actions is that between the occurrence of these two actions b was neither renamed nor zero-tested.

In general, what is required is to store in memory the cumulative effect of the zero tests and renaming in the part of the table not currently in view. To this end we associate with each rule $\tau \in \Delta$ a relation R_τ on Σ_ε according to the value of $op(\tau)$. The table below shows this association, where Id is the identity relation on Σ_ε.

$op(\tau)$	R_τ
$\sigma!, \sigma?$	Id
$zero(\sigma)$	$Id - \{(\sigma, \sigma)\}$
R	$R \cup \{\varepsilon, \varepsilon\}$

Additionally, the special symbol $-$ used for filling in the blank spaces is associated to the identity.

Now suppose that on row i of the padded computation table the sequence of transition rules is $\tau_1^i, \tau_2^i, \ldots, \tau_n^i$, and that τ_j^i is the transition currently in view. Then the sliding-window procedure stores in memory a pair of relations $\mathit{Left}_{i,j}$ and $\mathit{Right}_{i,j}$ on Σ, where $\mathit{Left}_{i,j} = R_{\tau_j^i} \circ \cdots \circ R_{\tau_1^i}$ and $\mathit{Right}_{i,j} = R_{\tau_n^i} \circ \cdots \circ R_{\tau_{j+1}^i}$. Note that $\mathit{Right}_{i,j}$ must be guessed since it refers to the part of the table to the right of the current view, which has not been seen yet. The correctness criterion on the current view is that if $\sigma, \sigma' \in \Sigma$ are vertically aligned, with $\sigma!$ on row i and $\sigma'?$ on row $i+1$, then $\sigma\ (\mathit{Left}_{i+1,j-1} \circ \mathit{Right}_{i,j})\ \sigma'$. Finally, observe that it is straightforward to verify the consistency of the guessed value of $\mathit{Right}_{i,j}$ from one view to the next.

Theorem 3. *The cycle-bounded reachability problem for CAROTs is solvable in polynomial space in the size of the channel automaton and polynomial space in the value of the cycle bound.*

Proof. Given a channel automaton $\mathcal{C}$ and a cycle bound N, the sliding-window procedure needs at any one time to store $2N$ binary relations on Σ_ε (the relations Left_i and Right_i described above, $i = 1, \ldots, N$). It needs also to remember the

contents of the $(3 \times N)$-window currently in view. Finally it needs to remember the left-most columnn of the computation table (to verify that it matches the right-most column). This all can be stored in polynomial space in $|\mathcal{C}|$ and in the value of N. Now, if $\mathcal{C}$ has an accepting execution, then it has one whose computation table has size less than $2 \cdot |\Delta|^N \cdot 2^{N \cdot (|\Sigma|+1)^2} + 1$. A non-deterministic, polynomial-space algorithm can thus guess the exponential sequence of configurations (i.e., list of N transitions) and check locally that they form a consistent computation of $\mathcal{C}$. □

Remark 4. Note that our algorithm could easily be adapted to cope with the cycle-bounded reachability of a full state (*i.e.*, location and content of the channel): it suffices to add transitions that would, from any point, run one more cycle of the channel and store its whole content in the location.

3 Metric Temporal Logic

3.1 Timed automata

Timed automata were introduced in [AD94] as an extension of finite automata with clocks. The transitions of a timed automaton are restricted by conditions, called guards, that are required to be fulfilled at the time transitions are fired. Given a finite set X of clocks, the set of guards $\Psi(X)$ is defined by the grammar

$$\varphi \ ::= \ \mathtt{tt} \ \mid \ \mathtt{ff} \ \mid \ \varphi \wedge \varphi \ \mid \ \varphi \vee \varphi \ \mid \ x \sim c$$

where $x \in X$, $\sim \in \{<, \leqslant, =, \geqslant, >\}$ and $c \in \mathbb{N}$. That a valuation $v \colon X \to \mathbb{R}^+$ satisfies a guard is defined by

$$v \models x \sim c \quad \Leftrightarrow \quad v(x) \sim c,$$

and with the obvious meaning for boolean operators.

We write $\mathbf{0}$ for the valuation mapping each clock to 0. Given a valuation v, a real t and a subset r of X, we define

- $v' = v + t$ is the valuation such that $v'(x) = v(x) + t$ for all clocks,
- $v' = [r \leftarrow 0]v$ is the valuation such that $v'(x) = 0$ if $x \in r$ and $v'(x) = v(x)$ for the other clocks.

Definition 5. *A* timed automaton *(TA) $\mathcal{A}$ is a tuple $(L, X, \Sigma, L_0, \delta)$ where L is a finite set of states, those in L_0 being initial, X is a finite set of clock variables, Σ is a finite alphabet of actions, and $\delta \subseteq L \times \Sigma \times \Psi(X) \times 2^X \times L$.*

Definition 6. *A timed word w is an infinite sequence $(\sigma_i, t_i)_{i \in \mathbb{N}}$ where $\sigma_i \in \Sigma$ and $t_i \in \mathbb{R}^+$ for each i, and such that the sequence $(t_i)_{i \in \mathbb{N}}$ is nondecreasing and diverges to infinity.*

Given a timed word $w = (w_i)_{i \in N}$ and an integer $j \in \mathbb{N}$, we write $w_{\geqslant j}$ for the subword $(w_{j+i})_{i \in \mathbb{N}}$.

Timed automata generate timed words in the following sense:

Definition 7. *Let $\mathcal{A} = (L, X, \Sigma, L_0, \delta)$ be a TA, and $w = (\sigma_i, t_i)_{i\in\mathbb{N}}$ be a timed word. We say that w is accepted by $\mathcal{A}$ if there exists a sequence $(\ell_i, v_i)_{i\in\mathbb{N}}$, with $\ell_i \in L$ and $v_i \colon X \to \mathbb{R}^+$, and such that*

- *$\ell_0 \in L_0$ and $v_0 = \mathbf{0}$;*
- *for each $i \geqslant 0$, there exists a transition $e_i = (\ell_i, \sigma_i, g_i, r_i, \ell_{i+1}) \in \delta$ such that $v_i + t_i - t_{i-1} \models g_i$ and $v_{i+1} = (v_i + t_i - t_{i-1})[r_i \leftarrow 0]$ (assuming $t_{-1} = 0$).*

3.2 Metric Temporal Logic

In this section we formally define the syntax and semantics of Metric Temporal Logic.

Definition 8. *The syntax of* Metric Temporal Logic (MTL) *[Koy90] is defined by the following grammar:*

$$\mathsf{MTL} \ni \varphi ::= \sigma \mid \neg\sigma \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathbf{U}_I\, \varphi \mid \varphi \,\widetilde{\mathbf{U}}_I\, \varphi$$

where σ ranges over a finite set of events Σ and I is an interval of $\mathbb{R}^+$ with bounds in $\mathbb{N} \cup \{+\infty\}$.

Following [Hen91,Hen98,HMP92,Wil94,AH93,AH94], among others, we interpret the logic over timed words.[5]

Definition 9. *Let $w = (\sigma_i, t_i)_{i\in\mathbb{N}}$ be an infinite timed word over Σ, and $k \in \mathbb{N}$. The (pointwise) semantics of* MTL *is defined recursively as follows (we omit Boolean operations):*

$$\begin{aligned}
w, k \models \sigma &\;\Leftrightarrow\; \sigma_k = \sigma \\
w, k \models \neg\sigma &\;\Leftrightarrow\; \sigma_k \neq \sigma \\
w, k \models \varphi \vee \psi &\;\Leftrightarrow\; w, k \models \varphi \text{ or } w, k \models \psi \\
w, k \models \varphi \wedge \psi &\;\Leftrightarrow\; w, k \models \varphi \text{ and } w, k \models \psi \\
w, k \models \varphi \,\mathbf{U}_I\, \psi &\;\Leftrightarrow\; \exists i > 0 \text{ s.t. } w, k+i \models \psi,\ t_{k+i} - t_k \in I \\
&\qquad\qquad \text{and } \forall 0 < j < i,\ w, k+j \models \varphi \\
w, k \models \varphi \,\widetilde{\mathbf{U}}_I\, \psi &\;\Leftrightarrow\; w, k \models \neg\Big((\neg\varphi) \,\mathbf{U}_I\, (\neg\psi)\Big).
\end{aligned}$$

If $w, 0 \models \varphi$, we write $w \models \varphi$.

[5] This is the so-called *pointwise* semantics. Another semantics, interval-based, is interpreted over continuous signals. See e.g. [Hen98,Ras99] for more details. As noted in [Hen98], the known complexity results for MITL hold both in the interval-based and in the pointwise semantics.

Additional operators, such as $\mathtt{tt}$ (true), $\mathtt{ff}$ (false), $\tilde{\wedge}$, $\Rightarrow$, $\Leftrightarrow$, $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{X}$, are defined in the usual way: $\mathbf{F}_I\,\varphi \equiv \mathtt{tt}\,\mathbf{U}_I\,\varphi$, $\mathbf{G}_I\,\varphi \equiv \mathtt{ff}\,\widetilde{\mathbf{U}}_I\,\varphi$, and $\mathbf{X}_I\,\varphi \equiv \mathtt{ff}\,\mathbf{U}_I\,\varphi$. We also use pseudo-arithmetic expressions to denote intervals. For example, '$= 1$' denotes the singleton $\{1\}$.

Let us point out that the main results of this paper also hold under a weakly monotonic semantics for time (in which the timestamps are merely nondecreasing), as well as under a non-strict semantics for temporal operators (in which the present time point is included).

Remark 10. Note that MTL is closed under negation, even though it does not appear in the grammar defining MTL.

3.3 Satisfiability and model checking

We consider the following two fundamental questions for MTL and various fragments thereof:

- The *satisfiability problem*, asking whether a given MTL formula φ is satisfiable, *i.e.*, whether $w \models \varphi$ for some infinite timed word w over Σ;
- The *model-checking problem*, asking whether a given timed automaton $\mathcal{A}$ satisfies a given MTL formula φ, *i.e.*, whether all timed words accepted by $\mathcal{A}$ satisfy φ. We write $\mathcal{A} \models \varphi$ when the answer is positive.

Among others, we identify the following syntactic fragments of MTL. *Linear Temporal Logic* (LTL) can be considered as the fragment of MTL in which modalities are not constrained (*i.e.*, where $\mathbb{R}^+$ is the only constraining interval). *Metric Interval Temporal Logic* (MITL) is the fragment of MTL where punctuality is not allowed (*i.e.*, where interval constraints are not singletons). Bounded-MTL is the fragment of MTL in which all interval constraints have finite length.

MITL was introduced in [AFH96], motivated by the role played by punctuality in the undecidability proof for MTL. The main result of [AFH96] was that model checking and satisfiability for MITL are EXPSPACE-Complete. The identification of Bounded-MTL as another decidable fragment of MTL is the first main result of the present paper. We show that both satisfiability and model checking for Bounded-MTL are EXPSPACE-Complete.

Example 11. Let the *variability* of a timed word be the maximum number of events that occur in any one time unit. We exhibit a family of Bounded-MTL formulas $\{\varphi_n\}_{n\in\mathbb{N}}$ such that the size of φ_n is linear in n, but the variability of any timed word satisfying φ_n is at least 2^{2^n}, *i.e.*, *doubly exponential* in n. We define $\varphi_n \equiv a \wedge \varphi_{Double} \wedge \mathbf{G}_{[0,2^n]}\,\varphi_{Double}$, where $\varphi_{Double} \equiv (a \rightarrow \mathbf{F}_{=1}\,(a \wedge \mathbf{X}_{\leqslant 1}\,b)) \wedge (b \rightarrow \mathbf{F}_{=1}\,(a \wedge \mathbf{X}_{\leqslant 1}\,b))$. If $\varrho \models \varphi_n$, then the variability of ϱ must (at least) double every time unit over the first 2^n time units.

Example 11 makes it surprising that the satisfiability problem for Bounded-MTL is in EXPSPACE. Indeed, most techniques used for timed model-checking rely on translating the formula into a timed automaton (e.g. MITL [AFH96] or

State-Clock Logic [RS97]). One would expect that a timed automaton equivalent to a given Bounded-MTL formula would, at any point in its run, have to remember (in its clock variables) all the events that happened in the previous time unit. But this would require space at least doubly exponential in the size of the formula.

Observe that while Bounded-MTL permits punctual formulas, it disallows unconstrained modalities. In particular, Bounded-MTL is not suitable to express invariance—the most basic type of temporal specification—and it does not subsume LTL (either syntactically or semantically). Intuitively, Bounded-MTL is only suitable for expressing time-bounded specifications. To remedy this deficiency we introduce Flat-MTL as the fragment of MTL generated by the grammar:

$$\text{Flat-MTL} \ni \varphi ::= \sigma \mid \neg\sigma \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathbf{U}_J\, \varphi \mid \underline{\psi}\, \mathbf{U}_I\, \varphi \mid \varphi\, \widetilde{\mathbf{U}}_J\, \varphi \mid \varphi\, \widetilde{\mathbf{U}}_I\, \underline{\psi}$$

where J ranges over the set of bounded intervals, I over the set of all intervals, and the underlined formula $\underline{\psi}$ ranges over LTL.

Notice immediately that Flat-MTL subsumes both LTL and Bounded-MTL, however it is not closed under negation. In fact, the most natural way to state our main results is in terms of the dual logic, which we call coFlat-MTL. This consists of the duals (*i.e.*, the negations) of Flat-MTL formulas. Correspondingly, the syntactic restriction determining coFlat-MTL as a subset of MTL is dual to that determining Flat-MTL: we require that, if I is unbounded, then formulas appearing on the right of $\mathbf{U}_I$ and on the left of $\widetilde{\mathbf{U}}_I$ be LTL formulas.

Like Flat-MTL, coFlat-MTL includes both LTL and Bounded-MTL. However, crucially, it is also closed under $\mathbf{G}_I$ for unbounded I, since $\mathbf{G}_I\, \varphi \equiv \bot\, \widetilde{\mathbf{U}}_I\, \varphi$. Thus we have the slogan:

$$\text{Bounded-MTL} + \text{Invariance} \;\subseteq\; \text{coFlat-MTL}\,.$$

This means that one can express a much more useful class of specifications in coFlat-MTL than in Bounded-MTL.

The main result of this paper is that the model-checking problem for coFlat-MTL is EXPSPACE-Complete. This last problem can be understood as a slight generalisation of the satisfiability problem for the dual logic Flat-MTL.

Example 12. The formula $\mathbf{G}\,(\mathit{request} \Rightarrow \mathbf{F}_{[0,1]}\,(\mathit{acquire} \wedge \mathbf{F}_{=1}\,\mathit{release}))$ says that every time the lock is requested, it is acquired within one time unit, and released after exactly one further time unit. This formula is in coFlat-MTL, but is not in Bounded-MTL (due to the unconstrained $\mathbf{G}$) and is not in MITL (due to the punctual $\mathbf{F}_{=1}$).

Given a timed automaton $\mathcal{A}$, to find a violation of the above formula one must search for a run of $\mathcal{A}$ such that after some *request*-event, every *acquire*-event in the subsequent time unit fails to be followed after exactly one time unit by a *release*-event. Intuitively, over a dense-time semantics, this task seems to require 'remembering' a potentially unbounded amount of information. Thus our EXPSPACE-Completeness result for model checking coFlat-MTL may appear surprising.

For comparison with previous work we describe one more fragment of MTL, called Safety-MTL. This is determined by the restriction that the Until modality only be constrained by bounded intervals:

$$\text{Safety-MTL} \ni \varphi ::= \sigma \mid \neg\sigma \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathbf{U}_I\, \varphi \mid \varphi \,\widetilde{\mathbf{U}}_J\, \varphi$$

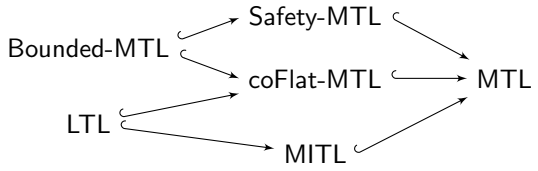
where I is required to be bounded and no restriction is placed on J. Informally Safety-MTL can express general invariance but only time-bounded eventuality.

Like coFlat-MTL, Safety-MTL includes Bounded-MTL and is closed under $\mathbf{G}$, that is,

$$\text{Bounded-MTL} + \text{Invariance} \;\subseteq \text{Safety-MTL}\,.$$

However, unlike coFlat-MTL, satisfiability is decidable for Safety-MTL whereas model checking is non-elementary.

We summarise the relationships between the various logics introduced above in the following diagram (where $\hookrightarrow$ indicates a syntactic inclusion):



Example 13. For later use, we also consider a weak inverse to the formula φ_{Double} introduced in Example 11, namely $\varphi_{Half} = \mathbf{F}_{=1}\, \mathtt{tt} \vee \mathbf{X}\, \mathbf{F}_{=1}\, \mathtt{tt}$. Now consider the formula

$$\varphi \equiv a \wedge \varphi_{Double} \wedge \mathbf{G}_{[0,2^n)}\, \varphi_{Double} \wedge \mathbf{G}_{[2^n,2^{n+1})}\, \varphi_{Half} \wedge \mathbf{F}_{=2^{n+1}}\, (a \wedge \mathbf{X}_{=1}\, \mathtt{tt})\,.$$

If $\varrho \models \varphi$, then, as explained in Example 11, ϱ has *at least* 2^{2^n} events in the time interval $[2^n, 2^n + 1)$. The point of φ is that it forces there to be *exactly* 2^{2^n} events in this time interval. Indeed, the conjunct $\mathbf{G}_{[2^n,2^{n+1})}\, \varphi_H$ ensures that for any integer i, $1 \leqslant i \leqslant 2^n$, the number of events in ϱ in the unit interval $[2^n + i, 2^n + i + 1)$ is at least $2^{2^n - i}$. Together with the requirement that there be a unique event in the time interval $[2^{n+1}, 2^{n+1} + 1)$, this forces the required property of exactly 2^{2^n} events in the time interval $[2^n, 2^n + 1)$.

Remark 14. Observe that the logics Flat-MTL and coFlat-MTL are not closed under negation: the formula $\mathbf{G}\, \mathbf{F}_{=1}\, a$ is in coFlat-MTL; its negation $\mathbf{F}\, \mathbf{G}_{=1}\, \neg a$ is not (at least syntactically) in coFlat-MTL.

3.4 Main results

Table 1 summarizes the complexity of the fragments of MTL defined above. Dark gray boxes correspond to results stated and proved elsewhere, whereas light gray boxes correspond to results that can be deduced straightforwardly from other papers. The undecidability of MTL is proved in [OW06a], while MITL

	Model Checking	**Satisfiability**
LTL	PSPACE-C.	PSPACE-C.
MITL	EXPSPACE-C.	EXPSPACE-C.
Bounded-MTL	EXPSPACE-C.	EXPSPACE-C.
Safety-MTL	Non-Prim.-Rec.	Non-Elem.
Flat-MTL	undec.	EXPSPACE-C.
coFlat-MTL	EXPSPACE-C.	Undec.
MTL	Undec.	Undec.

Table 1. Complexity of fragments of MTL (interpreted over infinite timed words)

and Safety-MTL have been defined and studied respectively in [AFH96] and in [OW05,OW06b].

In this paper, we state the following results:

- The model-checking problem for coFlat-MTL is in EXPSPACE (see sections 4, 5, and 6), which immediately implies the same result for Bounded-MTL.
- The model-checking and satisfiability problems for Bounded-MTL both are EXPSPACE-Hard (see section 7), which immediately implies that coFlat-MTL model checking is also EXPSPACE-Hard.
- Finally, the EXPSPACE complexity for Bounded-MTL reduces to PSPACE if constants are encoded in unary.

In addition, it is worth noticing that the undecidability proof of [OW06a] for the satisfiability of MTL over infinite words can also be used to prove that the satisfiability problem for coFlat-MTL (and thus the model-checking problem for Flat-MTL) is undecidable. The result that the satisfiability problem for Safety-MTL is non-elementary is a consequence of [OW05].

The proof that the model-checking of coFlat-MTL is in EXPSPACE can be sketched as follows:

- if φ is the formula that we want to verify, we first construct an alternating timed automaton (ATA) which recognizes all models of $\neg\varphi$ (Section 4.2);
- we then prove properties of that 1ATA (Section 4.3);
- we construct a CAROT which will simulate joint executions of the automaton we want to model-check and the above-mentioned 1ATA (Section 5);
- we gather everything to conclude (Section 6).

The EXPSPACE-Hardness proof for the satisfiability problem of Bounded-MTL is presented in section 7.

4 From MTL to Alternating Timed Automata

In this section, we recall the definition of alternating timed automata (ATA): a natural timed analog of alternating automata [LW05,OW05]. ATA generalise classical (Alur-Dill) timed automata [AD94] by allowing conjunctions in the transition relation, and, unlike the latter, are closed under complement. However language-emptiness is, in general, undecidable for ATA as universality in TA can be reduced to language-emptiness in ATA.

In this section, we present the model of ATA and give it a tree-based semantics. Then, for a given MTL formula φ, we construct an ATA $\mathcal{B}_\varphi$ that accepts all words satisfying φ. We show that if φ is in Flat-MTL then $\mathcal{B}_\varphi$ possesses certain structural properties, corresponding to the flatness of φ. Using these properties, we analyse the structure of the executions of $\mathcal{B}_\varphi$.

4.1 Alternating timed automata

Let L be a finite set of locations and X a finite set of clocks. We define $\Phi(L, X)$ as the set of formulas defined by the grammar:

$$\varphi \ ::= \ \mathtt{tt} \ \mid \ \mathtt{ff} \ \mid \ \varphi \wedge \varphi \ \mid \ \varphi \vee \varphi \ \mid \ \ell \ \mid \ x \sim c \ \mid \ x \cdot \varphi$$

where $\ell \in L$, $x \in X$ and $c \in \mathbb{N}$.

Let $M \in \mathbb{N}$ be an integer constant, aimed at being the maximal constant that appears in the automaton. For our purpose, the clocks will take their values in the set $\mathsf{Val} = [0, M] \cup \{\bot\}$, with value $\bot$ meaning that the value of the clock is strictly larger than M. We require that $\bot$ satisfies the following rules:

$$\bot + t = \bot \quad \text{for any } t \in \mathbb{R}^+ \qquad\qquad \bot > t \quad \text{for any } t \leqslant M$$
$$t + t' = \bot \quad \text{if } t + t' > M$$

Let $C = \{(\ell_i, v_i) \mid i \leqslant n\}$ be a set of states of $L \times \mathsf{Val}^X$, and $v \colon X \to \mathsf{Val}$ be a clock valuation. That C satisfies a formula $\varphi \in \Phi(L, X)$ under valuation v, denoted by $C \models_v \varphi$, is defined recursively as follows:

$$C \models_v \mathtt{tt} \qquad\qquad C \models_v \varphi_1 \wedge \varphi_2 \ \Leftrightarrow \ C \models_v \varphi_1 \text{ and } C \models_v \varphi_2$$
$$C \not\models_v \mathtt{ff} \qquad\qquad C \models_v \varphi_1 \vee \varphi_2 \ \Leftrightarrow \ C \models_v \varphi_1 \text{ or } C \models_v \varphi_2$$
$$C \models_v x \sim c \ \Leftrightarrow \ v(x) \sim c \qquad\qquad C \models_v \ell \ \Leftrightarrow \ (\ell, v) \in C$$
$$C \models_v x \cdot \varphi \ \Leftrightarrow \ C \models_{[x \leftarrow 0]v} \varphi$$

The set C is a *minimal model* of $\varphi \in \Phi(L, X)$ w.r.t. valuation v if $C \models_v \varphi$ and there is no proper subset $C' \subsetneq C$ such that $C' \models_v \varphi$.

Definition 15. *An* alternating timed automaton $\mathcal{A}$ *is a tuple* $(L, X, \Sigma, \delta_0, \delta, F)$ *where* L *is a finite set of locations,* X *is a finite set of clocks,* Σ *is a finite set of actions,* $\delta_0 \in \Phi(L, X)$ *is an initial condition,* $\delta \colon L \times \Sigma \to \Phi(L, X)$ *is the transition relation, and* $F \subseteq L$ *is a set of accepting locations.*

Definition 16. *Let S be a set of letters, and S^* be the set of finite words over S. A* tree *τ over S is a subset of S^* such that*

- *for any word $s_0\, s_1 \cdots s_p \in \tau$, we also have $s_0\, s_1 \cdots s_{p-1} \in \tau$;*
- *if $s_0 \cdots s_p$ and $s'_0 \cdots s'_q$ are two words of τ, then $s_0 = s'_0$. The word s_0 is the* root *of τ.*

An element of a tree is called a node*. Let $\nu = s_0 \cdots s_p$ be a node of a tree τ. The depth of ν is p; its label is s_p, and its set of labels of its successors is $\mathsf{succ}(\nu) = \{s' \in \Sigma \mid \nu \cdot s' \in \tau\}$. A* branch *of a tree is a maximal (finite or infinite) sequence of nodes $(\nu_i)_i$ such that ν_i is a prefix of ν_{i+1} for each i. The* subtree of τ rooted in ν *is the tree $\{s_p \cdot w \in S^* \mid \nu \cdot w \in \tau\}$. Last, a* forest *is a finite set of trees.*

Definition 17. *Given a timed alternating automaton $\mathcal{A}$ and a timed word $w = (\sigma_i, t_i)_{i\in\mathbb{N}}$, an* execution *of $\mathcal{A}$ over w is a forest $\{\tau_1, ..., \tau_k\}$ over the set $L \times \mathsf{Val}^X$, satisfying the following conditions:*

- *any root (ℓ, v) is such that $v = \mathbf{0}$, and the set of roots is a minimal model of the initial condition δ_0 under valuation $\mathbf{0}$;*
- *for each node ν of the forest with $\mathsf{depth}(\nu) = p$ and $\mathsf{label}(\nu) = (\ell, v)$, letting $v' = v + t_p - t_{p-1}$ (with $t_{-1} = 0$), the set $\mathsf{succ}(\nu)$ is a minimal model of $\delta(\ell, \sigma_p)$ under valuation v'.*

An execution is accepting if every infinite branch in the forest contains infinitely many nodes whose labels are in $F \times \mathsf{Val}^X$.

It is easily observed that any execution forest is finitely branching, since we require each set of successors to be a minimal model of the transition formula. Still, this notion of minimality is *local*. Following [EJ91], we define the notion of *memoryless execution forest* as follows:

Definition 18. *An execution forest of $\mathcal{A}$ is* memoryless *if, for any two identically labelled nodes ν_1 and ν_2 appearing at the same depth i, the respective subtrees rooted at ν_1 and ν_2 are identical.*

This restriction is no loss of generality:

Proposition 19. *Any accepting execution forest can be made memoryless (while preserving acceptance).*

Proof. We adapt the proof of [EJ91, Theorem 4.4]. Let $\{\tau_1, ..., \tau_n\}$ be an accepting forest. With each node ν of this forest, we associate an integer $\mu(\nu)$ representing the maximal distance to the next accepting state:

$$\begin{aligned} &\mu(\nu) = 0 && \text{if } \mathsf{label}(\nu) \in F \times \mathsf{Val}^X \\ &\mu(\nu) = 1 + \max\{\mu(\nu \cdot s) \mid s \in L \times \mathsf{Val}^X\} && \text{otherwise.} \end{aligned}$$

In particular, $\mu(\nu)$ is $-\infty$ if ν has no successor. Since the execution forest is finitely branching and accepting, we have $\mu(\nu) < +\infty$ for any node in the forest.

Also, clearly enough, if a node ν is not accepting, then $\mu(\nu)$ is strictly larger than the value of μ at any immediate successor of ν.

Now, we recursively define a sequence of forests $(f_j)_{j\in\mathbb{N}}$, starting with $f_0 = \{\tau_1, ..., \tau_n\}$. The forest f_{j+1} is defined from f_j as follows: for any (maximal) set of equally labelled nodes ν, ν', ..., $\nu^{(n)}$ appearing at depth j in f_j, we pick one of those states having the least μ, and replace each subtree of f_j rooted at those nodes with the subtree rooted at the selected node. The resulting forest is f_{j+1}.

This defines a Cauchy sequence of forests for the standard distance on forests (the distance between two forests is 2^{-d} where d is the minimal depth at which both forests differ), and we write f_∞ for the limit of that sequence. It is easily seen that f_∞ is still an execution forest. Now, if ν is a node of f_j at depth less than j, the same node appears again in f_{j+k} for any $k \geqslant 0$, and, by construction of the sequence of forests, the value of μ at that node is nonincreasing. In the end, the value of μ at any node ν of depth k of f_∞ is at most the value of μ at the same node in f_k, and is thus finite (or $-\infty$). As an easy consequence, there cannot exist an infinite branch in f_∞ containing only finitely many nodes labelled with accepting states. □

In the sequel, we transform MTL formulas into alternating automata involving only one clock:

Definition 20. *A* one-clock alternating timed automaton*, 1ATA for short, is an alternating timed automaton with a single clock ($|X| = 1$). Labels of nodes in an execution of a1ATA can thus be seen as pairs in $L \times \mathsf{Val}$.*

4.2 From MTL to ATAs

Following the lines of the untimed case, MTL formulas can be transformed into "equivalent" 1ATA [LW05,OW05]. We describe this construciton here, and study some of its properties in the next section.

Let $\varphi \in \mathsf{MTL}$, with maximal constant M. The set of *restricted subformulas* of φ is the set of formulas of MTL defined recursively as follows:

- $\mathrm{SubF}(p) = \{p\}$;
- $\mathrm{SubF}(\neg p) = \{\neg p\}$;
- $\mathrm{SubF}(\psi_1 \vee \psi_2) = \mathrm{SubF}(\psi_1) \cup \mathrm{SubF}(\psi_2)$;
- $\mathrm{SubF}(\psi_1 \wedge \psi_2) = \mathrm{SubF}(\psi_1) \cup \mathrm{SubF}(\psi_2)$;
- $\mathrm{SubF}(\psi_1 \,\mathbf{U}_I\, \psi_2) = \{\psi_1 \,\mathbf{U}_I\, \psi_2\} \cup \mathrm{SubF}(\psi_1) \cup \mathrm{SubF}(\psi_2)$;
- $\mathrm{SubF}(\psi_1 \,\widetilde{\mathbf{U}}_I\, \psi_2) = \{\psi_1 \,\widetilde{\mathbf{U}}_I\, \psi_2\} \cup \mathrm{SubF}(\psi_1) \cup \mathrm{SubF}(\psi_2)$.

With φ, we associate the 1ATA $\mathcal{B}_\varphi = (L, \{x\}, 2^{\mathsf{AP}}, \delta_0, \delta, F)$ as follows:

- $L = \mathrm{SubF}(\varphi)$;
- $\delta_0 = \varphi$, where conjunctions and disjunctions are part of δ_0 (*i.e.*, φ is not one state of the automaton but a positive boolean combinations of states);

– the transition relation δ is defined as follows:

$$\begin{array}{rlrl} \delta(p,\sigma) = \mathtt{tt} & \text{if } p \in \sigma & \delta(p,\sigma) = \mathtt{ff} & \text{if } p \notin \sigma \\ \delta(\neg p,\sigma) = \mathtt{ff} & \text{if } p \in \sigma & \delta(\neg p,\sigma) = \mathtt{tt} & \text{if } p \notin \sigma \end{array}$$

$$\begin{aligned} \delta(\psi_1 \,\mathbf{U}_I\, \psi_2, \sigma) &= \Big((\psi_1 \,\mathbf{U}_I\, \psi_2) \wedge (x \cdot \psi_1)\Big) \vee \Big((x \in I) \wedge (x \cdot \psi_2)\Big) \\ \delta(\psi_1 \,\widetilde{\mathbf{U}}_I\, \psi_2, \sigma) &= \Big((\psi_1 \,\widetilde{\mathbf{U}}_I\, \psi_2) \vee (x \cdot \psi_1)\Big) \wedge \Big((x \notin I) \vee (x \cdot \psi_2)\Big) \end{aligned}$$

– F is the set of $\widetilde{\mathbf{U}}$-locations.

The following lemma entails the correction of our construction. In this lemma, we use the notation $w, i \models^v \psi$, with $v \in \mathsf{Val}$ with the following meaning:

$$\begin{array}{rcll} w,i \models^v \psi_1 \,\mathbf{U}_I\, \psi_2 & \Leftrightarrow & w,i \models \psi_1 \,\mathbf{U}_{I-v}\, \psi_2 & \text{if } v \in [0,M] \\ w,i \models^v \psi_1 \,\widetilde{\mathbf{U}}_I\, \psi_2 & \Leftrightarrow & w,i \models \psi_1 \,\widetilde{\mathbf{U}}_{I-v}\, \psi_2 & \text{if } v \in [0,M] \\ w,i \models^\infty \psi_1 \,\mathbf{U}_I\, \psi_2 & \Leftrightarrow & w,i \models \psi_1 \,\mathbf{U}\, \psi_2 & \text{and } I \text{ is unbounded} \\ w,i \models^\infty \psi_1 \,\widetilde{\mathbf{U}}_I\, \psi_2 & \Leftrightarrow & w,i \models \psi_1 \,\widetilde{\mathbf{U}}\, \psi_2 & \text{or } I \text{ is bounded} \\ w,i \models^v \psi' & \Leftrightarrow & w,i \models \psi' & \text{for other kinds of formulas} \end{array}$$

where $I - x = \{y \mid x + y \in I\}$. Note that $w, i \models^0 \psi$ is equivalent to $w, i \models \psi$.

Lemma 21. *Let $\psi \in SubF(\varphi)$, $w \in (\Sigma \times \mathbb{R}^+)^\omega$, $i \in \mathbb{Z}^+$ and $v \in \mathsf{Val}$. There exists an accepting execution tree on input word $w_{\geqslant i}$ with root (ψ, v) iff $w, i \models^v \psi$.*

We omit the proof of this result, as it is an easy adaptation of the inductive proof of [OW07, Prop. 6.4].

The automaton $\mathcal{B}_\varphi$ enjoys several important properties:

***linearity*:** there is a linear order $\leqslant$ on L such that for all $\ell \in L$ and $\sigma \in \Sigma$, each location ℓ' occurring in $\delta(\ell, \sigma)$ satisfies $\ell' \leqslant \ell$. This property is a classical property when transforming a formula of some linear-time temporal logic into an alternating automaton.

***locality*:** ℓ is the only location that can occur in $\delta(\ell, \sigma)$ not under the scope of a reset '$x.$'. Also, ℓ never occurs in $\delta(\ell, \sigma)$ under the scope of a reset. This is because timing constraints are bound to modalities (contrary to TPTL [AH94], which involves formula clocks).

When φ is in Flat-MTL, then $\mathcal{B}_\varphi$ also enjoys the following property:

***flatness*:** there is a subset of (untimed) locations $L_u \subseteq L$ such that $\ell \in L_u$ implies that $\delta(\ell, \sigma)$ contains no clock constraints. Furthermore, for all $\ell \in L$, $\delta(\ell, \sigma)$ has the form $((x \leqslant c) \wedge \varphi_1) \vee \varphi_2 \vee \varphi_3$, and $\varphi_2 \in \Phi(\{\ell\} \cup L_u, x)$ whereas φ_3 does not mention ℓ. This condition can be read as follows: after a certain amount of time, location ℓ cannot make a simultaneous transition to itself and another location in $L \setminus L_u$.

4.3 Ranking Flat-MTL

In this section, we analyse the structure of the execution forests of those 1ATA arising from Flat-MTL formulas. Roughly speaking, the main result of this section, Theorem 24, says that the segments of such an execution forest in which the automaton clocks are *active* have a short total duration. Here we say that a clock (value) is active if it is no greater than the maximum clock constant M of the automaton, otherwise we say that it is inactive. By extension, we also say that a state (ℓ, v) with $\ell \notin \mathsf{LTL}$ is active or inactive according to whether v is active or inactive. If $\ell \in \mathsf{LTL}$, then a state (ℓ, v) is always said inactive.

In the rest of this section let $\mathcal{B}_\varphi$ denote an ATA arising from a Flat-MTL formula φ, and let M be the maximum clock constant of $\mathcal{B}_\varphi$. Recall that the set of locations of $\mathcal{B}_\varphi$ is the set $\mathrm{SubF}(\varphi)$ of modal subformulas of φ.

Given an execution forest of $\mathcal{B}_\varphi$, its i-th configuration is the set of states labelling the nodes at depth i. An execution forest of an ATA thus generates a sequence of configurations $\varrho\colon C_0 \to C_1 \to \ldots \to C_k \to \ldots$ Next we define a rank function on configurations based on the distinction between active and inactive states.

Let $<$ be a linear order on $\mathrm{SubF}(\varphi)$ such that $\varphi_1 < \varphi_2$ whenever φ_1 is a subformula of φ_2 (one such can always be chosen). Furthermore, let Γ denote the set $\mathrm{SubF}(\varphi) \times \{\bot, \top\}$ ordered by $(\varphi, \mathrm{I}) < (\varphi', \mathrm{I}')$ iff $\varphi < \varphi'$ or $\varphi = \varphi'$ and $\mathrm{I} = \bot$ and $\mathrm{I}' = \top$. We think of $\top$ as representing an active clock, whereas (following the notation introduced in Section 4.1) $\bot$ denotes an inactive clock.

Definition 22. *Given a configuration C of $\mathcal{B}_\varphi$, let the non-*LTL *formulas occurring in C be written $\{\varphi_i\}_{i=1}^k$, where $\varphi_k > \varphi_{k-1} > \ldots > \varphi_1$. If none of the φ_i is paired with an active clock in C, then we define* $\mathsf{rank}(C)$ *to be the word $(\varphi_k, \bot)\ldots(\varphi_2, \bot)(\varphi_1, \bot)$. Otherwise, let φ_j be the maximum among all formulas appearing in C that are paired with an active clock, and define* $\mathsf{rank}(C)$ *to be the word $(\varphi_k, \bot)\ldots(\varphi_{j+1}, \bot)(\varphi_j, \top)$. We order the ranks of configurations according to the lexicographic order on Γ^*, denoted $\preceq$.*

Example 23. Let the maximum clock constant in φ be $M = 3$ and let $C = \{(\varphi_1, 2.4), (\varphi_1, \bot), (\varphi_2, 0.8), (\varphi_2, \bot), (\varphi_3, \bot), (\varphi_4, \bot)\}$ be a configuration of $\mathcal{B}_\varphi$, where $\varphi_4 > \varphi_3 > \varphi_2 > \varphi_1$. Then $\mathsf{rank}(C) = (\varphi_4, \bot)(\varphi_3, \bot)(\varphi_2, \top)$, that is, we record the maximum active state and all inactive states above it.

Let $\varrho\colon C_0 \to C_1 \to \ldots \to C_k \to \ldots$ be a sequence of configurations in a run of $\mathcal{B}_\varphi$ on the timed word $(\sigma_i, t_i)_{i\in\mathbb{N}}$. If I is an integer interval (possibly unbounded), write $\varrho[I]\colon C_i \to C_{i+1} \to \ldots \to C_j \to \ldots$ for the segment of ϱ composed of all C_i's with $i \in I$ ($\varrho[I]$ may be finite or infinite). Furthermore, define the *active duration* of $\varrho[I]$, denoted $\mathsf{duration}(\varrho[I])$, to be 0 if none of the C_i's with $i \in I$ contains an active clock, and $\sup\{t_j - t_i + M \mid i, j \in I\}$ otherwise. Intuitively, $\mathsf{duration}(\varrho[I])$ gives an upper bound for the amount of time that an active clock is present in the segment of ϱ with positions in I. (If I is bounded and j is the maximal value in I, this segment includes the time delay between positions j and $j+1$ in ϱ, hence the extra term M in the expression for $\mathsf{duration}(\varrho[I])$.)

Theorem 24. *Let $\varrho\colon C_0 \to C_1 \to \ldots \to C_k \to \ldots$ be the sequence*[6] *of configurations of a memoryless run of $\mathcal{B}_\varphi$. Then there is a partition $\mathcal{I}$ of $\mathbb{N}$ into at most $\lfloor\varphi\rfloor \cdot 2^{\lfloor\varphi\rfloor}$ intervals, where for each interval I in $\mathcal{I}$, $\varrho[I]$ has active duration at most $2M+1$, the last interval is (unbounded and) fully inactive, and $\lfloor\varphi\rfloor$ is the number of non-*LTL *modal subformulas of φ.*

Proof. Define an equivalence $\equiv$ on $\mathbb{N}$ by $n \equiv m$ iff $\mathsf{rank}(C_n) = \mathsf{rank}(C_m)$. Now Lemma 25 (below) says that rank is non-increasing along ϱ; it follows that the equivalence classes of $\equiv$ are intervals. Furthermore, the index of the equivalence relation is bounded by the number of ranks, which is easily seen to be no more than $\lfloor\varphi\rfloor \cdot 2^{\lfloor\varphi\rfloor}$. Finally, it follows from Lemma 26 (below) that the active duration of any equivalence class is at most $2M+1$. □

It remains to prove the two technical lemmas quoted in the proof of Theorem 24.

Lemma 25. *If $\varrho\colon C_0 \to C_1 \to \ldots \to C_k \to \ldots$ is the sequence of configurations in a memoryless run of $\mathcal{B}_\varphi$, then $\mathsf{rank}(C_{i+1}) \preceq \mathsf{rank}(C_i)$ for each $i \in \mathbb{N}$.*

Proof. We split the transition from C_i to C_{i+1} into two steps: a time-elapse step, where each clock in C_i increases by some fixed amount, and a discrete step, where state changes are performed according to the transition function of $\mathcal{B}_\varphi$. We show that neither of these steps is rank-increasing.

For the time-elapse step, observe that for any configuration C and time delay $t \in \mathbb{R}_+$, $\mathsf{rank}(C+t) \preceq \mathsf{rank}(C)$, where $C + t = \{(\psi, v+t) : (\psi, v) \in C\}$. This is because the only possible difference between C and $C+t$ is that active clocks in C may become inactive in $C+t$; but this cannot increase the rank (reflecting the fact that $\bot < \top$).

Write δ for the transition function of $\mathcal{B}_\varphi$, and let $\sigma \in \Sigma$. For the discrete step, suppose that $C = \{(\psi_i, v_i)\}_{i\in I}$ is a configuration and that $C' = \bigcup_i D_i$, where D_i is a minimal model of $\delta(\psi_i, \sigma)$ with respect to v_i for each $i \in I$.[7] Furthermore, for a contradiction, suppose that $\mathsf{rank}(C) \prec \mathsf{rank}(C')$, with $\gamma \in \Gamma$ the letter in $\mathsf{rank}(C')$ occurring in the first position in which $\mathsf{rank}(C)$ and $\mathsf{rank}(C')$ differ. Since the letters in $\mathsf{rank}(C)$ appear in descending order we can assume that γ does not appear in $\mathsf{rank}(C)$ at all. We consider two cases according to whether γ is inactive or active.

The first case is that $\gamma = (\psi, \bot)$ for some $\psi \in \mathrm{SubF}(\varphi) \setminus \mathsf{LTL}$. Then there exists $i \in I$ such that $(\psi, \bot) \in D_i$. By locality of $\mathcal{B}_\varphi$ (cf. Section 4.2), we must have $\psi_i = \psi$ and $v_i = \bot$. Thus $\gamma = (\psi, \bot)$ appears in $\mathsf{rank}(C)$, contradicting the assumption on γ.

The second case is that $\gamma = (\psi, \top)$ for some $\psi \in \mathrm{SubF}(\varphi) \setminus \mathsf{LTL}$. Then there exists $i \in I$ and a clock value $v \leqslant M$ such that $(\psi, v) \in D_i$. By linearity of $\mathcal{B}_\varphi$

[6] If the run happens to be finite, then we still assume the sequence of configurations be infinite, possibly ending by empty configurations.

[7] Since ϱ is memoryless, the set of states at each configuration in ϱ can always be calculated from the set of states of the previous configuration in this manner.

we have $\psi \leqslant \psi_i$; if also $v_i \leqslant M$ then some active state at least as high as $(\psi, \top)$ appears in $\mathsf{rank}(C)$. Since $\mathsf{rank}(C)$ and $\mathsf{rank}(C')$ agree on all letters higher than γ, we must also have that $(\psi_i, \top)$ appears in $\mathsf{rank}(C')$, which is impossible. Thus we may assume that $v_i = \bot$. But then, since $v_i \neq v$, by locality of $\mathcal{B}_\varphi$ it must hold that $\psi < \psi_i$, and by flatness of $\mathcal{B}_\varphi$, we have that ψ_i does not appear in D_i. (Flatness dictates that ψ and ψ_i cannot both appear in D_i.) In fact, we can conclude that $(\psi_i, \bot)$ does not appear in C' (by locality of $\mathcal{B}_\varphi$ it cannot appear in D_j for $j \neq i$). But then $(\psi_i, \bot)$ does not appear in $\mathsf{rank}(C')$ and $(\psi_i, \bot) > \gamma$, contradicting the assumption on γ. □

Lemma 26. *Suppose $\varrho\colon C_0 \to C_1 \to \ldots \to C_k \to \ldots$ is the sequence of configurations in a memoryless run of $\mathcal{B}_\varphi$ on a timed word $(\sigma_i, t_i)_{i\in\mathbb{N}}$. If C_i is active, $j > i$ and $t_j - t_i > M$, then $\mathsf{rank}(C_j) \prec \mathsf{rank}(C_i)$.*

Proof. Write $\mathsf{rank}(C_i) = (\varphi_k, \bot)\ldots(\varphi_2, \bot)(\varphi_1, \top)$ and suppose, for a contradiction, that $\mathsf{rank}(C_i) = \mathsf{rank}(C_{i+1}) = \ldots = \mathsf{rank}(C_j)$. In particular, for $2 \leqslant p \leqslant k$, the node $(\varphi_p, \bot)$ is present in each of the configurations $C_i, C_{i+1}, \ldots, C_j$. This means that in the execution tree underlying ϱ, between depths i and j, any node labelled $(\varphi_p, \bot)$, for $2 \leqslant p \leqslant k$, also has a child labelled $(\varphi_p, \bot)$ (by locality of $\mathcal{B}_\varphi$, no state can make a (discrete) transition to $(\varphi_p, \bot)$ apart from $(\varphi_p, \bot)$ itself). By flatness of $\mathcal{B}_\varphi$ we conclude that the only possible depth-j descendents of a depth-i node labelled $(\varphi_p, \bot)$, $2 \leqslant p \leqslant k$, are also labelled by $(\varphi_p, \bot)$, or by LTL formulas.

Now, since $\mathsf{rank}(C_i) = \mathsf{rank}(C_j)$, $(\varphi_1, \top)$ occurs in $\mathsf{rank}(C_j)$. Thus there is a state $(\varphi_1, v) \in C_j$ such that $v \leqslant M$. From the above argument, the depth-i ancestor of this state can only be labelled (φ_1, u) for some u. Since $t_j - t_i > M$, the clock x is reset somewhere on the path from (φ_1, u) to (φ_1, v). But this contradicts linearity and locality of $\mathcal{B}_\varphi$, since these conditions imply that any clock reset on a path must be accompanied by a strict reduction in the rank of the locations along the path. □

Remark 27. The ranking argument can be strengthened if we restrict to $\mathsf{Bounded\text{-}MTL}$ formulas: defining $\mathsf{rank}'(C)$ as being only made of the highest active subformula (or being empty is no formula is active), we can easily prove that Lemmas 25 and 26 still hold with this definition of the rank, and Theorem 24 is strengthened as the number of possible ranks is now $\lfloor\varphi\rfloor$.

5 From ATAs to CAROTs

In this section, we define a simulation of ATAs by CAROTs. This roughly corresponds to the powerset construction used for transforming an (untimed) alternating automaton into a non-deterministic automaton [MH84], except that we cannot bound the size of a configuration in the timed case due to the presence of clock variables. Instead we use the channel to store encodings of configurations, which are levels in a run tree of the ATA being simulated. In this simulation, the cycling of the channel corresponds to the evolution of time, and global renaming and occurrence testing are used to simulate discrete transitions of the ATA.

Using Theorem 24, we show that an ATA $\mathcal{B}_\varphi$ corresponding to a Flat-MTL formula φ can be simulated by a cycle-bounded CAROT. Then we use Theorem 3, concerning the cycle-bounded reachability problem for CAROTs, to prove an EXPSPACE upper bound for model checking.

We first fix some notations: let $\mathcal{A} = (L_\mathcal{A}, X_\mathcal{A}, \Sigma, L^0_\mathcal{A}, \delta_\mathcal{A})$ be the timed automaton under study, and $\mathcal{B} = (L_\mathcal{B}, \Sigma, \delta^0_\mathcal{B}, \delta_\mathcal{B}, F_\mathcal{B})$ be the 1ATA corresponding to $\neg\varphi$ (previously called $\mathcal{B}_{\neg\varphi}$) constructed in the previous section. We call $x_\mathcal{B}$ its single clock.

We note $\mathsf{REG} = \{0, 1, \ldots, M, \bot\}$ where M is the maximal constant appearing in $\mathcal{A}$ or in $\mathcal{B}$. If $\gamma \in \mathbb{R}^+$ and $\gamma \leqslant M$, we write $\mathsf{reg}(\gamma)$ for the largest integer in REG which is smaller than or equal to γ. We write $\mathsf{reg}(\bot) = \bot$. We also define the following two sets:

$$S = (L_\mathcal{B} \times \{x_\mathcal{B}\} \times \mathsf{Val}) \cup (L_\mathcal{A} \times X_\mathcal{A} \times \mathsf{Val})$$
$$R = (L_\mathcal{B} \times \{x_\mathcal{B}\} \times \mathsf{REG}) \cup (L_\mathcal{A} \times X_\mathcal{A} \times \mathsf{REG})$$

and their sets of subsets $V = \wp(S)$ and $\Lambda = \wp(R)$.

A joint $\mathcal{A}/\mathcal{B}$-configuration is composed of a state (ℓ, v) of $\mathcal{A}$ with $\ell \in L_\mathcal{A}$, $v\colon X_\mathcal{A} \to \mathsf{Val}$ and a finite set of states (ℓ_i, v_i) of $\mathcal{B}$, with $\ell_i \in L_\mathcal{B}$ and $v_i \in \mathsf{Val}$ for $i \in I$. Such a configuration C can be written as the element $\{(\ell_i, x_\mathcal{B}, v_i) \mid i \in I\} \cup \{(\ell, x, v(x)) \mid x \in X_\mathcal{A}\}$ of V.

Now given a configuration C, we partition C into a sequence of subsets $C_0, C_1, \ldots, C_n, C_\bot$, such that $C_\bot = \{(\ell, x, v) \in C \mid \ell \in \mathsf{LTL} \text{ or } v = \bot\}$, $\bigcup_{i=0}^n C_i = C \setminus C_\bot$, and if $i, j \neq \bot$, for all $(\ell, x, v) \in C_i$ and $(\ell', x', v') \in C_j$, $\mathit{frac}(v) \leqslant \mathit{frac}(v')$ iff $i \leqslant j$ (so that (ℓ, x, v) and (ℓ', x', v') are in the same block C_i iff v and v' have the same fractional part). We assume in addition that the fractional part of elements in C_0 is 0 (even if it means that $C_0 = \varnothing$). Note that $C_\bot$ contains all inactive and LTL formulas of the configuration (following the vocabulary of the previous section). We then let $H(C) = \mathsf{reg}(C_0)\mathsf{reg}(C_1)\mathsf{reg}(C_2)\ldots\mathsf{reg}(C_n)\mathsf{reg}(C_\bot) \in \Lambda^*$. In the following, we remove the superfluous $x_\mathcal{B}$'s and $\bot$'s in the letters, in order to ease readability.

The joint $\mathcal{A}/\mathcal{B}$-behaviour is then composed of transitions $C \xrightarrow{\sigma} C'$ for $\sigma \in \Sigma$ and $C \xrightarrow{t} C'$ for $t \in \mathbb{R}^+$ in the usual way. Using the abstraction function, it is possible to define a discrete transition system which abstracts away precise timing information, but which simulates joint $\mathcal{A}/\mathcal{B}$-behaviours:

Lemma 28 (Bisimulation lemma [OW07]). *The equivalence relation over configurations defined by 'C ≡ C′ iff H(C) = H(C′)' is a bisimulation relation.*[8]

Example 29. Consider for instance a configuration C encoded by the word

$$H(C) = \{(\ell_0, 2), (\ell, x, 3)\} \cdot \{(\ell, y, 1)\} \cdot \{(\ell_1, 3), (\ell_2, 1)\} \cdot \{(\ell, z), \ell_3\}.$$

We assume that the maximal constant is 4. The encoding of the successor of C is obtained by cycling around the letters (except the last one) of the word (and

[8] We restrict the transition relation for joint $\mathcal{A}/\mathcal{B}$-behaviours to memoryless behaviours.

increasing the values of the regions accordingly). Thus the first delay successor of $H(C)$ is

$$\varnothing \cdot \{(\ell_0, 2), (\ell, x, 3)\} \cdot \{(\ell, y, 1)\} \cdot \{(\ell_1, 3), (\ell_2, 1)\} \cdot \{(\ell, z), \ell_3\}$$

(all states with integral values are just above the integer), the next successor is

$$\{(\ell_1, 4), (\ell_2, 2)\} \cdot \{(\ell_0, 2), (\ell, x, 3)\}\{(\ell, y, 1)\} \cdot \{(\ell, z), \ell_3\}$$

(the states with maximal fractional part reach the next integer), the next one is

$$\varnothing \cdot \{(\ell_2, 2)\} \cdot \{(\ell_0, 2), (\ell, x, 3)\} \cdot \{(\ell, y, 1)\} \cdot \{(\ell, z), \ell_3, \ell_1\}$$

as the state ℓ_1 is now over the maximal constant 4. Simulating discrete transitions is easy as it only consists in applying the transition rules of $\mathcal{A}$ and $\mathcal{B}$ to all states of the word (see above-mentioned references for more details).

We will take advantage of this discrete abstraction to define a CAROT $\mathcal{C}$ which will 'recognize' the discrete joint $\mathcal{A}/\mathcal{B}$-behaviours. In order to simplify the construction we will use shortcuts for actions like "$\langle\lambda\rangle!$" (resp. "$\langle\lambda\rangle?$") where $\lambda \subseteq \Lambda$ which will mean that we successively apply the actions "$\rangle!$" (resp. "$\rangle?$"), "$\alpha!$" (resp. "$\alpha?$") for every $\alpha \in \lambda$ (the order doesn't matter) and finally "$\langle!$" (resp. "$\langle?$"). We will also write $\bigwedge_{i\in I} R(\alpha_i, \varepsilon)$ to mean that every α_i for $i \in I$ is removed from the channel.

The channel will be used to store the 'unbounded' part of the information, namely the successive configurations of $\mathcal{B}$. Since $\mathcal{A}$ must synchronize with $\mathcal{B}$, its timing information will also be stored on the channel. The discrete states of the CAROT will store only a bounded amount of information, namely the location of $\mathcal{A}$, the region it lies in, the set of clocks of $\mathcal{A}$ having integer values, and the $\mathcal{B}$-part of the sets $\mathsf{reg}(C_0)$ and $\mathsf{reg}(C_\perp)$. For instance, a configuration C such that

$$H(C) = \{(\ell_1, r_0), (\ell_2, r_4), (\ell, x, r_2)\} \cdot \{(\ell, y, r_1), (\ell_1, r_5), (\ell_2, r_3)\} \cdot \{(\ell, z, r_7)\} \cdot \{\ell_3\}$$

is encoded by the discrete information

$$\Big(\ell, \varrho, \{x\}, \{(\ell_1, r_0), (\ell_2, r_4)\}, \{\ell_3\}\Big)$$

where $\varrho(x) = r_2$, $\varrho(y) = r_1$ and $\varrho(z) = r_7$, and by the channel content (where we read from the left):

	$\langle$	z	$\rangle$	$\langle$	(ℓ_2,r_3)	(ℓ_1,r_5)	y	$\rangle$	

We construct the CAROT $\mathcal{C} = (Q, q_0, \Gamma, \Delta)$ (without accepting conditions for the moment) as follows:

- the alphabet Γ is the union of $L_\mathcal{B} \times \mathsf{REG} \setminus \{\perp\}$, the set of clocks $X_\mathcal{A}$, and the set of two brackets $\{\langle, \rangle\}$;

– we define $\mathcal{Q}$ the product set $L_{\mathcal{A}} \times \mathsf{REG}^{X_{\mathcal{A}}} \times \wp(X_{\mathcal{A}}) \times \wp(L_{\mathcal{B}} \times (\mathsf{REG} \setminus \{\bot\})) \times \wp(L_{\mathcal{B}})$. An element of $\mathcal{Q}$ contains the following information:
  - the current location of $\mathcal{A}$,
  - the integral part of the clocks of $\mathcal{A}$,
  - the set of clocks of $\mathcal{A}$ whose fractional part is zero,
  - the set of states $(\ell_{\mathcal{B}}, x_{\mathcal{B}})$ of the current configuration of $\mathcal{B}$ in which the clock $x_{\mathcal{B}}$ is an integer,
  - the set of inactive (or LTL) formulas in the current configuration of $\mathcal{B}$.

  Then, the set of states Q of $\mathcal{C}$ is the union of the following components:
  - $\mathcal{Q}$ is the set of 'regular' states,
  - $\mathcal{Q} \times \Sigma$ has an extra information which is the label of the transition which is being taken. It will be used when simulating a move,
  - $\mathcal{Q} \times (\Sigma \times \wp(L_{\mathcal{B}} \times \mathsf{REG}))$, which contains additional information on the configuration of $\mathcal{B}$ stored on the channel.
– the initial states of the CAROT are those of the form $(\ell_{\mathcal{A}}, \{0\}^{X_{\mathcal{A}}}, X_{\mathcal{A}}, \lambda_{\mathcal{B}}, \kappa_{\mathcal{B}})$ with $\ell_{\mathcal{A}} \in L^0_{\mathcal{A}}$, $\lambda_{\mathcal{B}} \subseteq L_{\mathcal{B}} \times \{0\}$, $\kappa_{\mathcal{B}} \subseteq L_{\mathcal{B}} \cap \mathsf{LTL}$, and $(\lambda_{\mathcal{B}} \cup \kappa_{\mathcal{B}}) \models_0 \delta^0_{\mathcal{B}}$;
– the transition relation Δ is again presented as a union of several kinds of transitions:

(1) $\Big\{(\ell_{\mathcal{A}}, r_{\mathcal{A}}, z_{\mathcal{A}}, \lambda_{\mathcal{B}}, \kappa_{\mathcal{B}}) \xrightarrow{\langle z_{\mathcal{A}} \cup \lambda'_{\mathcal{B}} \rangle !} (\ell_{\mathcal{A}}, r_{\mathcal{A}}, \varnothing, \varnothing, \kappa'_{\mathcal{B}}) \;\Big|\; z_{\mathcal{A}} \cup \lambda_{\mathcal{B}} \neq \varnothing,\ \lambda'_{\mathcal{B}} = \lambda_{\mathcal{B}} \setminus (L_{\mathcal{B}} \times \{M\}) \text{ and } \kappa'_{\mathcal{B}} = \kappa_{\mathcal{B}} \cup \{\ell \mid (\ell, M) \in \lambda_{\mathcal{B}}\}\Big\}$: these transitions represent time elapsing from a configuration where the configurations of $\mathcal{A}$ or $\mathcal{B}$ (possibly both) contain a clock with fractional part zero. As time elapses, the fractional part becomes positive, and those clocks are then pushed on the channel.

(2) $\Big\{(\ell_{\mathcal{A}}, r_{\mathcal{A}}, \varnothing, \varnothing, \kappa_{\mathcal{B}}) \xrightarrow{\langle z_{\mathcal{A}} \cup \lambda_{\mathcal{B}} \rangle ?} (\ell_{\mathcal{A}}, r'_{\mathcal{A}}, z_{\mathcal{A}}, \lambda'_{\mathcal{B}}, \kappa_{\mathcal{B}}) \;\Big|\; r'_{\mathcal{A}}(x) = \begin{cases} r_{\mathcal{A}}(x) & \text{if } x \notin z_{\mathcal{A}} \\ r_{\mathcal{A}}(x) + 1 & \text{if } x \in z_{\mathcal{A}} \end{cases} \text{ and } \lambda'_{\mathcal{B}} = \{(\ell, i+1) \mid (\ell, i) \in \lambda_{\mathcal{B}}\}\Big\}$: these ones are the symmetric case of the previous one: if the values of some of the clocks become integer due to time elapse, those clocks are read out of the channel and stored in the discrete location of the CAROT.

(3) $\Big\{(\ell_{\mathcal{A}}, r_{\mathcal{A}}, z_{\mathcal{A}}, \lambda_{\mathcal{B}}, \kappa_{\mathcal{B}}) \xrightarrow{R(Y,\varepsilon)} \big((\ell'_{\mathcal{A}}, r'_{\mathcal{A}}, z'_{\mathcal{A}}, \lambda_{\mathcal{B}}, \kappa_{\mathcal{B}}), \sigma\big) \;\Big|\; \ell_{\mathcal{A}} \xrightarrow{g,\sigma,Y} \ell'_{\mathcal{A}},\ (r_{\mathcal{A}}, z_{\mathcal{A}}) \models g,\ r'_{\mathcal{A}} = r_{\mathcal{A}}[Y \leftarrow 0], \text{ and } z'_{\mathcal{A}} = z_{\mathcal{A}} \cup Y\Big\}$: those transitions are the first step in the encoding of the synchronized transition of $\mathcal{A}$ and $\mathcal{B}$. This part encodes the modification of the configuration of $\mathcal{A}$ when firing a transition $(\ell_{\mathcal{A}}, g, \sigma, Y, \ell'_{\mathcal{A}})$. It stores the letter σ for 'synchronizing' with the transitions of $\mathcal{B}$.

(3′) $\Big\{\big((\ell'_{\mathcal{A}}, r'_{\mathcal{A}}, z'_{\mathcal{A}}, \lambda_{\mathcal{B}}, \kappa_{\mathcal{B}}), \sigma\big) \xrightarrow{zero(\overline{\lambda_{\mathrm{ch}}})} \big((\ell'_{\mathcal{A}}, r'_{\mathcal{A}}, z'_{\mathcal{A}}, \lambda_{\mathcal{B}}, \kappa_{\mathcal{B}}), (\sigma, \lambda_{\mathrm{ch}})\big) \;\Big|\; \lambda_{\mathrm{ch}} \subseteq L_{\mathcal{B}} \times \mathsf{REG} \setminus \{\bot\}\Big\}$: these transitions store a set λ_{ch} in the location. This set is a superset of the content of the channel (as checked by $zero(\overline{\lambda_{\mathrm{ch}}})$).

$$(3'')\ \left\{\left((\ell'_{\mathcal{A}}, r'_{\mathcal{A}}, z'_{\mathcal{A}}, \lambda_{\mathcal{B}}, \kappa_{\mathcal{B}}), (\sigma, \lambda_{\mathrm{ch}})\right) \xrightarrow{R(\lambda^0_{\mathrm{ch}}, \varepsilon)} (\ell'_{\mathcal{A}}, r'_{\mathcal{A}}, z'_{\mathcal{A}}, \lambda^{\mathrm{new}}_{\mathcal{B}}, \kappa^{\mathrm{new}}_{\mathcal{B}}) \ \middle|\ \lambda^0_{\mathrm{ch}} \subseteq \lambda_{\mathrm{ch}} \text{ and } \left(\lambda^{\mathrm{new}}_{\mathcal{B}} \cup \kappa^{\mathrm{new}}_{\mathcal{B}}\right) \cup \left(\lambda_{\mathrm{ch}} \setminus \lambda^0_{\mathrm{ch}}\right)^+ \models \varphi_{\mathcal{B}}\right\}$$

where

$$\varphi_{\mathcal{B}} = \bigwedge_{(\ell,\alpha)\in\lambda_{\mathcal{B}}} \left[\delta(\ell,\sigma) \wedge \text{“}x \in \alpha\text{”}\right] \wedge \bigwedge_{(\ell,\alpha)\in\lambda_{\mathrm{ch}}} \left[\delta(\ell,\sigma) \wedge \text{“}x \in \alpha^+\text{”}\right] \wedge \bigwedge_{\ell\in\kappa_{\mathcal{B}}} \delta(\ell,\sigma)\,.$$

These transitions select a subset λ^0_{ch} of clocks that will be reset, and a set $\lambda^{\mathrm{new}}_{\mathcal{B}} \cup \kappa^{\mathrm{new}}_{\mathcal{B}}$ of new states of the configuration of $\mathcal{B}$, and check that those states satisfy the transition condition. In this transition relation, α^+ represents the open region $(\alpha, \alpha+1)$. This notation is extended to λ_{ch} in the obvious way.

Cases (3), (3′) and (3″) of the above construction are aimed at being glued altogether and at being viewed as a single step. So does case (1), and so does case (2). We write $(\mathrm{disc}_1, \mathrm{ch}_1) \Rightarrow (\mathrm{disc}_2, \mathrm{ch}_2)$ for a single step of the CAROT. A pair $(\mathrm{disc}, \mathrm{ch})$ such that $(\mathrm{disc}, \mathrm{ch})$ is a configuration of $\mathcal{C}$ corresponds to a single encoding of configurations into a word of Λ^* (as explained earlier in the section), and *vice-versa*. If C is a joint $\mathcal{A}/\mathcal{B}$-configuration, we write $d(C)$ for the corresponding discrete state of the CAROT, and $c(C)$ for the corresponding content of the channel. In the following, we write $H(C) \cong (d(C), c(C))$ (as there is a one-to-one correspondence between these two encodings). Moreover, given two words $h = \lambda_1 \dots \lambda_n$ and $h' = \lambda'_1 \dots \lambda'_m$ in Λ^*, we say that h is a subword of h' whenever there is an injection $\iota : \{1, \dots, n\} \to \{1, \dots m\}$ such that for every $i \in \{1, \dots, n\}$, $\lambda_i \subseteq \lambda'_{\iota(i)}$.

Then the correctness of the CAROT can be stated as follows:

Lemma 30. *The relation* $\prec \ \subseteq \Lambda^* \times \left(\mathcal{Q} \times \Gamma^*\right)$ *defined as*

$$w \prec (d, c) \Leftrightarrow w \cong (d, c)$$

is a simulation relation (for the “single-step” transition $\Rightarrow$ *of the CAROT).*

Conversely, the relation $\lhd \ \subseteq \left(\mathcal{Q} \times \Gamma^*\right) \times \Lambda^*$ *defined as*

$$(d, c) \lhd w \Leftrightarrow w \text{ subword of } h \text{ where } h \cong (d, c)$$

is a simulation relation.

Proof. The first property is an obvious consequence of the construction.

The second property requires more attention. Assume $\left(d_1, c_1\right) \Rightarrow (d_2, c_2)$ in $\mathcal{C}$. We distinguish several cases:

– either it is a delay transition (rule (1) or (2)), and in that case, for every w_1 subword of h_1 where $h_1 \cong (d_1, c_1)$, there exists w_2 such that $w_1 \to w_2$ is a delay transition, and w_2 is a subword of h_2 if $h_2 \cong (d_2, c_2)$;

– or it is a sequence of the rules (3), (3′) and (3″). Assume that w_1 is a subword of $h_1 \cong (d_1, c_1)$. Rule (3) precisely mimics a σ-move in automaton $\mathcal{A}$. Then, rule (3′) overapproximates the content of the channel (and thus overapproximates c_1), guessing it is λ_{ch} (thus, $c_{1|\mathcal{B}} \subseteq \lambda_{\text{ch}}$ and w_1 is even less constraining). Then, under this assumption, rule (3″) computes a successor following the rules of $\mathcal{B}$. Thus there exists w_2 such that $w_1 \xrightarrow{\sigma} w_2$ and w_2 is a subword of $h_2 \cong (d_2, c_2)$. □

Thus, we can deduce properties of joint $\mathcal{A}/\mathcal{B}$-behaviours from properties of executions of the CAROT. However, the CAROT $\mathcal{C}$ does not take into account the original accepting condition of $\mathcal{B}$. We thus slightly modify the above construction and apply a Miyano-Hayashi construction [MH84] to take into account the accepting condition of $\mathcal{B}$ (thanks to Theorem 24, we know we can only restrict to the discrete information given by the states of the CAROT (inactive or LTL formulas)). This construction for untimed alternating automata is also described in [Var96, Prop. 20]. We modify the CAROT as follows: the component $\kappa_{\mathcal{B}} \in \wp(L_{\mathcal{B}})$ is replaced by a pair $(\kappa_{\mathcal{B},1}, \kappa_{\mathcal{B},2}) \in \wp(L_{\mathcal{B}}) \times \wp(L_{\mathcal{B}})$ such that $\kappa_{\mathcal{B}} = \kappa_{\mathcal{B},1} \cup \kappa_{\mathcal{B},2}$. Intuitively, $\kappa_{\mathcal{B},2}$ will store states that have been created by branches of the forest that recently hit a state of F, the set of repeated locations of $\mathcal{B}$, whereas $\kappa_{\mathcal{B},1}$ will store the other ones. We just need to change the transition rules of $\mathcal{C}$, and in particular the rules of types (1), (2) and (3″).

The set (1) is changed by setting $\kappa'_{\mathcal{B},1} = \kappa_{\mathcal{B},1} \cup \{\ell \mid (\ell, M) \in \lambda_{\mathcal{B}} \text{ and } \ell \in F\}$ and $\kappa'_{\mathcal{B},2} = \kappa_{\mathcal{B},2} \cup \{\ell \mid (\ell, M) \in \lambda_{\mathcal{B}} \text{ and } \ell \notin F\}$. Also, we remove from the sets (1) and (2) the transitions that go out of a state in which $\kappa_{\mathcal{B},1} = \varnothing$. From those states, transition (4), defined as

(4) $(\ell_{\mathcal{A}}, r_{\mathcal{A}}, z_{\mathcal{A}}, \lambda_{\mathcal{B}}, (\varnothing, \kappa_{\mathcal{B},2})) \to (\ell_{\mathcal{A}}, r_{\mathcal{A}}, z_{\mathcal{A}}, \lambda_{\mathcal{B}}, (\kappa_{\mathcal{B},2}, \varnothing))$.

The set (3″) is then replaced by the following set:

$$
\begin{aligned}
(3'')\ \Big\{ \Big((\ell'_{\mathcal{A}}, r'_{\mathcal{A}}, z'_{\mathcal{A}}, \lambda_{\mathcal{B}}, (\kappa_{\mathcal{B},1}, \kappa_{\mathcal{B},2})), (p, \lambda_{\text{ch}}) \Big) &\xrightarrow{R(\lambda^0_{\text{ch}}, \varepsilon)} \\
&(\ell'_{\mathcal{A}}, r'_{\mathcal{A}}, z'_{\mathcal{A}}, \lambda^{\text{new}}_{\mathcal{B}}, (\kappa^{\text{new}}_{\mathcal{B},1}, \kappa^{\text{new}}_{\mathcal{B},2})) \ \Big| \\
&\exists X, Y \subseteq L_{\mathcal{B}} \text{ s.t. } \lambda^0_{\text{ch}} \subseteq \lambda_{\text{ch}}, \Big(\lambda^{\text{new}}_{\mathcal{B}} \cup X\Big) \cup \Big(\lambda_{\text{ch}} \setminus \lambda^0_{\text{ch}}\Big)^+ \models \varphi_{\mathcal{B},1}, \\
&\Big(\lambda^{\text{new}}_{\mathcal{B}} \cup Y\Big) \cup \Big(\lambda_{\text{ch}} \setminus \lambda^0_{\text{ch}}\Big)^+ \models \varphi_{\mathcal{B},2}, \\
&\kappa^{\text{new}}_{\mathcal{B},1} = X \setminus F, \text{ and } \kappa^{\text{new}}_{\mathcal{B},2} = Y \cup (X \cap F) \Big\}
\end{aligned}
$$

where

$$\varphi_{\mathcal{B},i} = \bigwedge_{(\ell,\alpha)\in\lambda_{\mathcal{B}}} \Big[\delta(\ell,\sigma) \wedge \text{“}x \in \alpha\text{”}\Big] \wedge \bigwedge_{(\ell,\alpha)\in\lambda_{\text{ch}}} \Big[\delta(\ell,\sigma) \wedge \text{“}x \in \alpha^+\text{”}\Big] \wedge \bigwedge_{\ell\in\kappa_{\mathcal{B},i}} \delta(\ell,\sigma).$$

The set of repeated states of this new CAROT, denoted $\mathcal{C}_{\mathcal{A},\neg\varphi}$, is then the set of states of the form $(\ell_{\mathcal{A}}, r_{\mathcal{A}}, z_{\mathcal{A}}, \lambda_{\mathcal{B}}, (\varnothing, \kappa_{\mathcal{B},2}))$

This is a slight adaptation of Miyano-Hayashi construction [MH84] to CAROTs, and as a matter of fact, it is not difficult to prove that it is correct in the following sense:

Proposition 31. *There is an accepting run ϱ in $\mathcal{C}_{\mathcal{A},\neg\varphi}$ iff $\mathcal{A} \not\models \varphi$.*

Proof. First assume that $\mathcal{A} \not\models \varphi$. Then there exists $w \in \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}_{\neg\varphi})$. From Prop. 19, there is a memoryless execution of $\mathcal{B}_{\neg\varphi}$ accepting w, thus a sequence of configurations that can be partitioned as in Theorem 24. Also, there is an execution of $\mathcal{A}$ on input word w. It is easily checked that this yields an execution of the CAROT, which we must show is accepting.

From the partition of Theorem 24, we know that the execution is either finite (thus accepting), or eventually reaches a state from which all configurations of $\mathcal{B}_{\neg\varphi}$ are inactive. From that point on, from any node in the execution tree of $\mathcal{B}_{\neg\varphi}$ on w, the distance to the next accepting state is finite. Thus, from any configuration, after a finite number of states, it is possible to end up in a state of the CAROT where $\kappa_{\mathcal{B},1}$ is empty. This provides us with an accepting execution of $\mathcal{C}_{\mathcal{A},\neg\varphi}$.

Conversely, from an accepting execution of $\mathcal{C}_{\mathcal{A},\neg\varphi}$, there must be an infinite number of transitions of type 3, in order to fulfill the acceptance condition. We thus get a timed word w by gathering the sequence of letters that appear as second item of the states in $\mathcal{Q} \times \Sigma$. By construction of the CAROT, this timed word belongs to $\mathcal{L}(\mathcal{A})$. We also obtain a sequence of configurations of $\mathcal{B}_{\neg\varphi}$, which corresponds to an execution forest of $\mathcal{B}_{\neg\varphi}$. Between any two visits of a state where $\kappa_{\mathcal{B},1}$ is empty, all the branches of the execution forest must have encountered an accepting state, which entails that $\mathcal{B}_{\neg\varphi}$ accepts w. □

Remark 32. Note that we did only the construction for 1ATA corresponding to MTL formulas, but this method could be used to simulate with a CAROT any product of a timed automaton with a 1ATA (using the original renaming operator of the model, and not only the deletion operator).

6 Combining All Together

6.1 Membership in EXPSPACE

Proposition 33. *Let $\mathcal{A}$ be a timed automaton with set of clocks X, and $\varphi \in$ coFlat-MTL. Let $\mathcal{C}_{\mathcal{A},\neg\varphi}$ denote the CAROT that simulates joint executions of $\mathcal{A}$ and $\mathcal{B}_{\neg\varphi}$, as described above. Then, $\mathcal{A} \not\models \varphi$ iff there is an infinite computation ϱ of $\mathcal{C}_{\mathcal{A},\neg\varphi}$ such that we can write ϱ as $\varrho' \cdot \varrho''$ where:*

- *the number of cycles of the channel during ϱ' is bounded by an exponential in the sizes of φ and $\mathcal{A}$,*
- *the only information stored on the channel along ϱ'' are clocks of $\mathcal{A}$ (the size of the channel is then bounded by $|X|$),*
- *the Büchi condition of $\mathcal{C}_{\mathcal{A},\neg\varphi}$ is satisfied along ϱ''.*

Proof. Applying Theorem 24, we decompose ϱ as a sequence of alternatively active and inactive runs. Then ϱ'' will be the last (inactive) one, while ϱ' is the concatenation of the other ones. Thus ϱ' contains active parts whose durations sum up to at most $(2M+1) \cdot \lfloor \varphi \rfloor \cdot 2^{\lfloor \varphi \rfloor}$, and at most $\lfloor \varphi \rfloor \cdot 2^{\lfloor \varphi \rfloor}$ inactive parts. In each of these inactive runs, we can assume that each region of $\mathcal{A}$ is visited at most once. Their durations are thus bounded by the number of regions of $\mathcal{A}$, which is exponential and can be written in polynomial space. Hence the bound on the duration of ϱ'. The statements concerning ϱ'' are straightforward, since ϱ'' involves only inactive formulas, which are stored in the discrete state of the CAROT. □

Finally, we state our main result:

Theorem 34. *The model-checking problem for* coFlat-MTL *(and* Bounded-MTL*) is in EXPSPACE.*

Proof. The procedure runs in two steps:

- the first step, corresponding to part ϱ' of Proposition 33, consists in running the algorithm of Theorem 3 on-the-fly;
- the second step corresponds to ϱ'', when the size of the channel is bounded. This case can thus be reduced to simulating a finite-state automaton.

We now detail both steps. The first is an adaptation of the algorithm proposed in Section 2, which we run on-the-fly (*i.e.*, without explicitely computing $\mathcal{C}_{\mathcal{A},\neg\varphi}$, which would require doubly-exponential space). It consists in storing only the sliding window (and the relations $Left_i$ and $Right_i$, which are small), and, at each step, guessing the next sliding window and checking that each line corresponds to a transition of the CAROT. The height of the window is the total duration of ϱ'. It is made of at most $\lfloor \varphi \rfloor \cdot 2^{\lfloor \varphi \rfloor}$ active or inactive parts. An active part has duration at most $2M+1$. On the other hand, inactive parts can be shortened to duration at most $r_{\mathcal{A}} \cdot 2^{\lfloor \varphi \rfloor}$, where $r_{\mathcal{A}}$ is the number of regions of $\mathcal{A}$, by removing cycles. Thus, the size of the sliding window is at most $3 \times \left[(2M+1) + (r_{\mathcal{A}} \cdot 2^{\lfloor \varphi \rfloor})\right] \cdot \lfloor \varphi \rfloor \cdot 2^{\lfloor \varphi \rfloor}$. Now, each state of the CAROT has size exponential in $|\mathcal{A}|$ and $\lfloor \varphi \rfloor$, so that exponential space is sufficient for storing the full sliding window. Now, guessing the next sliding window and checking that it corresponds to a sequence (one for each line of the window) of transitions of the CAROT can be achieved in exponential space also.

That way, we are able to compute a reachable state of the CAROT in exponential space. It remains to prove that the "bounded" part, corresponding to ϱ'', corresponds to an accepting tail of the computation of the CAROT. Since the channel is bounded, this computation can be achieved in EXPSPACE by keeping track of the exact content of the channel. □

Our algorithm can easily be adapted in order to solve satisfiability:

Corollary 35. *The satisfiability problem for* Flat-MTL *(and* Bounded-MTL*) is in EXPSPACE.*

Following Remark 27, we can get the better bound of $(2M+1) \cdot \lfloor \varphi \rfloor$ for the total duration of ϱ' in the proof of Proposition 33. This lowers the complexity of Bounded-MTL model-checking to PSPACE under the assumption that constants are encoded in unary.

Corollary 36. *The model-checking and satisfiability problems are in PSPACE for* Bounded-MTL *with constants encoded in unary.*

7 Hardness Proof

In this subsection we show that the satisfiability problem for Bounded-MTL is EXPSPACE-hard. EXPSPACE-Hardness easily follows for the satisfiability of Flat-MTL and the model checking of Flat-MTL.

Theorem 37. *The satisfiability problem for* Bounded-MTL *is EXPSPACE-Hard.*

Proof. Let $\mathcal{M}$ be a 2^n-space-bounded Turing machine. Given an input X to $\mathcal{M}$, we construct in logarithmic space a Bounded-MTL formula φ_X that is satisfiable if and only if $\mathcal{M}$ accepts X. To some extent, this reduction follows a familiar pattern (see [AH93,AH94]): we encode computations of $\mathcal{M}$ as timed words by associating configurations of $\mathcal{M}$ with sequences of timed events; then punctual formulas of the kind $p \to \mathbf{F}_{=1}\, q$ are used to copy the encoding of a configuration in one time unit into the subsequent time unit.

However there are two significant differences between our proof and related hardness proofs for real-time logics and automata. Firstly we encode a whole computation history of $\mathcal{M}$ in a single time unit rather than encoding one configuration per time unit as in the works cited above. Indeed, the latter encoding would only allow us to prove EXPTIME-Hardness for Bounded-MTL. A second novel feature of our proof is the way we ensure an exact correspondence between events in successive time units. Note that while the formula $\mathbf{G}\,(a \to \mathbf{F}_{=1}\, b)$ ensures that every a-event is followed one time unit later by a b-event, it is not possible in general (without using past connectives) to require that every b-event be preceded one time unit earlier by an a-event [AH93,AH94]. However, as we explain below, by employing an idea from Example 13, we establish the required one-to-one correspondence between events in successive time units.

Next we define the formula φ_X and explain its components in detail. Suppose that $\mathcal{M}$ has set of control states Q and tape alphabet Γ ($\triangleright \in \Gamma$); then the alphabet of events used by $\varphi_{\mathcal{M}}$ is $\{a, b\} \cup \Sigma \cup \dot{\Sigma}$, where $\Sigma = \{\gamma, \gamma_q : \gamma \in \Gamma, q \in Q\}$ and $\dot{\Sigma} = \{\dot{\sigma} : \sigma \in \Sigma\}$. Intuitively the event γ represents a tape cell that currently contains $\gamma \in \Gamma$, whereas the event γ_q represents a tape cell that currently contains γ and is pointed to by the read head of $\mathcal{M}$, while $\mathcal{M}$ is in control state Q. The meaning of the dot superscript will be explained later.

The formula φ_X is a conjunction:

$$\varphi_X \equiv a \wedge \varphi_D \wedge \mathbf{G}_{I_0}\, \varphi_D \wedge \mathbf{G}_{I_1}\, \varphi_{GUESS} \wedge \mathbf{G}_{I_2}\, \varphi_{CHECK} \wedge \mathbf{G}_{I_3}\, \varphi_H \wedge \mathbf{F}_{I_4}\, (a \wedge \mathbf{X}_{=1}\, \mathtt{tt})$$

where φ_D and φ_H are the formulas of Examples 11 and 13.

Let $n = |X|$. The intervals $I_0, \ldots, I_4$, which are contiguous, are defined by $I_0 = [0, 2^n + n)$, $I_1 = [2^n + n, 2^n + n + 1)$, $I_2 = [2^n + n + 1, 2^{n+1} + n + 1)$, $I_3 = [2^{n+1} + n + 1, 2^{n+2} + 2n + 1)$ and $I_4 = \{2^{n+2} + 2n + 1\}$. We think of these intervals as corresponding to various phases in our simulation of $\mathcal{M}$ on input X. The phases corresponding to intervals I_0 and I_3 follow the pattern of Examples 11 and 13. They respectively involve repeated doubling and repeated halving of variability. Given that φ_{GUESS} and φ_{CHECK} guarantee that every event in the interval $I_1 \cup I_2$ is followed by an event one time unit later, a similar analysis to Example 13 shows that there are exactly $2^{2^n} 2^n$ events in each unit-length time interval $[k, k+1)$ for $k \in I_1 \cup I_2$. In turn this allows φ_{GUESS} and φ_{CHECK} to determine a one-to-one correspondence between events in successive time intervals via formulas of the type $\sigma_1 \rightarrow \mathbf{F}_{=1}\, \sigma_2$.

Intuitively, the role of $\mathbf{G}_{I_1}\, \varphi_{GUESS}$ is to guess a computation history of $\mathcal{M}$; thus we have $\varphi_{GUESS} \equiv (a \vee b) \rightarrow \mathbf{F}_{=1}\, (\bigvee_{\sigma \in \Sigma} \sigma)$. Given that $\varphi_D \wedge \mathbf{G}_{I_0}\, \varphi_D$ produces an alternating sequence of $2^n 2^{2^n}$ a- and b-events in the time interval $[2^n + n, 2^n + n + 1)$, the effect of $\mathbf{G}_{I_1}\, \varphi_{GUESS}$ is to write a sequence of events, say w_{hist}, of length $2^n 2^{2^n}$ in the time interval $[2^n + n + 1, 2^n + n + 2)$.

The role of $\mathbf{G}_{I_2}\, \varphi_{CHECK}$ is first to check that w_{hist} matches the regular expression $(\triangleright \dot{\Sigma} \Sigma^{2^n - 2})^{2^{2^n}}$, and furthermore that w_{hist} encodes a computation history of $\mathcal{M}$ on input X. The definition of φ_{CHECK} is such that it causes w_{hist} to be copied from one unit-length time interval to the next unit-length time interval, with the caveat that the superscripted dots move one place to the right in each successive interval. For instance, φ_{CHECK} contains the formula $\dot{\sigma_1} \wedge \mathbf{X}\, \sigma_2 \rightarrow \mathbf{F}_{=1}\, \sigma_1 \wedge \mathbf{X}\, \mathbf{F}_{=1}\, \dot{\sigma_2}$ as a conjunct. Finally, after 2^n time units, the sequence of events in the time interval $[2^{n+1} + n + 1, 2^{n+1} + n + 2)$ should match the regular expression $(\triangleright \Sigma^{2^n - 2} \dot{\Sigma})^{2^{2^n}}$.

Intuitively, the dot indicates the tape cell that is currently being verified by φ_{CHECK}. The part of φ_{CHECK} that verifies that the indicated tape cell is correctly updated from one configuration to the next is expressed in LTL. For example, if $\mathcal{M}$ in control state q can move its read head to the right without writing a new symbol, then we include

$$(\dot{\gamma}_q \wedge \mathbf{X}\, \delta) \rightarrow (\neg \dot{\Sigma}\, \mathcal{U}\, (\gamma \wedge \mathbf{X}\, \dot{\delta}_r)$$

as a component of φ_{CHECK}. Thus φ_{CHECK} verifies the correctness of the i-th tape cell in each configuration in a single time unit, and, in 2^n time units, over the course of the interval I_2, it verifies the correctness of w_{hist}. □

Note that a very similar proof can be used to get that the satisfiability problem for Bounded-MTL is PSPACE-Hard when constants are encoded in unary.

8 Conclusion

In this paper, we have proposed the logic coFlat-MTL as a counterpart to MITL, until now considered to be the only linear-time timed temporal logic having

reasonable complexity. Although both logics are incomparably expressive, coFlat-MTL allows most specifications that are interesting in practice, whilst retaining punctuality. Moreover, its model-checking problem exhibits no cost over that of MITL. As specifications tend to be relatively small, we feel justified in considering the complexity of model checking coFlat-MTL to be feasible, at least in theory. The real test will consist in applying our results in practice.

References

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AFH96] R. Alur, T. Feder, and Th. A. Henzinger. The benefits of relaxing punctuality. *Journal ACM*, 43(1):116–146, 1996.

[AH92] R. Alur and Th. A. Henzinger. Logics and models of Real-Time: a survey. In *Real-Time: Theory in Practice, Proc. REX Workshop 1991*, LNCS 600, p. 74–106. Springer, 1992.

[AH93] R. Alur and Th. A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[AH94] R. Alur and Th. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[EJ91] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proc. 32nd Annual Symp. on Found. of Computer Science (FOCS'91)*, p. 368–377. IEEE Comp. Soc. Press, 1991.

[Hen91] Th. A. Henzinger. *The Temporal Specification and Verification of Real-Time Systems*. PhD thesis, Stanford University, CA, USA, 1991.

[Hen98] Th. A. Henzinger. It's about time: Real-time logics reviewed. In *Proc. 9th Int. Conf. Concurr. Theory (CONCUR'98)*, LNCS 1466, p. 439–454. Springer, 1998.

[HMP92] Th. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proc. 19th Int. Coll. on Automata, Languages and Programming (ICALP'92)*, LNCS 623, p. 545–558. Springer, 1992.

[HR04] Y. Hirshfeld and A. M. Rabinovich. Logics for real time: Decidability and complexity. *Fundamenta Informaticae*, 62(1):1–28, 2004.

[Koy90] R. Koymans. Specifying real-time properties with Metric Temporal Logic. *Real-Time Systems*, 2(4):255–299, 1990.

[LW05] S. Lasota and I. Walukiewicz. Alternating timed automata. In *Proc. 8th Int. Conf. Found. of Software Science and Computation Structures (FoSSaCS'05)*, LNCS 3441, p. 250–265. Springer, 2005.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on omega-words. *Theoretical Computer Science*, 32:321–330, 1984.

[MNP06] O. Maler, D. Nickovic, and A. Pnueli. From MITL to timed automata. In *Proc. 4th Int. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, LNCS 4202, p. 274–189. Springer, 2006.

[OW05] J. Ouaknine and J. Worrell. On the decidability of Metric Temporal Logic. In *Proc. 19th Annual Symp. on Logic in Computer Science (LICS'05)*, p. 188–197. IEEE Comp. Soc. Press, 2005.

[OW06a] J. Ouaknine and J. Worrell. On Metric Temporal Logic and faulty Turing machines. In *Proc. 9th Int. Conf. Found. of Software Science and Computation Structures (FoSSaCS'06)*, LNCS 3921, p. 217–230. Springer, 2006.

[OW06b] J. Ouaknine and J. Worrell. Safety Metric Temporal Logic is fully decidable. In *Proc. 12th Int. Conf. Tools and Algorithms for the Constr. and Analysis of Systems (TACAS'06)*, LNCS 3920, p. 411–425. Springer, 2006.

[OW07] J. Ouaknine and J. Worrell. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Computer Science*, 2007. Submitted.

[Ras99] J.-F. Raskin. *Logics, Automata and Classical Theories for Deciding Real Time.* PhD thesis, Université de Namur, Belgium, 1999.

[RS97] J.-F. Raskin and P.-Y. Schobbens. State clock logic: a decidable real-time logic. In *Proc. Int. Workshop on Hybrid and Real-Time Systems (HART'97)*, LNCS 1201, p. 33–47. Springer, 1997.

[Var96] M. Y. Vardi. An automata-theoretic approach to Linear Temporal Logic. In *Proc. Logics for Concurr.: Structure versus Automata*, LNCS 1043, p. 238–266. Springer, 1996.

[Wil94] Th. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Proc. 3rd Int. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, LNCS 863, p. 694–715. Springer, 1994.