On Metric Temporal Logic and Faulty Turing Machines

Joël Ouaknine and James Worrell

Oxford University Computing Laboratory, UK
`{joel,jbw}@comlab.ox.ac.uk`

Abstract. Metric Temporal Logic (MTL) is a real-time extension of Linear Temporal Logic that was proposed fifteen years ago and has since been extensively studied. Since the early 1990s, it has been widely believed that some very small fragments of MTL are undecidable (i.e., have undecidable satisfiability and model-checking problems). We recently showed that, on the contrary, some substantial and important fragments of MTL are decidable [19, 20]. However, until now the question of the decidability of full MTL over infinite timed words remained open.
In this paper, we settle the question negatively. The proof of undecidability relies on a surprisingly strong connection between MTL and a particular class of faulty Turing machines, namely 'insertion channel machines with emptiness-testing'.

1 Introduction

The theory of automated verification in the untimed world has by now achieved a respectable maturity: there is a plethora of modelling and specification formalisms, with well-understood associated algorithms—see, e.g., [22] for a comprehensive survey of the field. Over the past two decades, many researchers have attempted to extend this methodology to the real-time world, in which *quantitative* timing constraints are of interest. One of the earliest and most prominent real-time specification formalisms to be proposed was Metric Temporal Logic (MTL) [16, 6], which extends Linear Temporal Logic (LTL) in that the various temporal operators are annotated with time intervals. For example, whereas the LTL formula $\Box(\mathit{req} \implies \Diamond \mathit{grant})$ specifies that every *req* is always eventually followed by a *grant*, the MTL formula $\Box(\mathit{req} \implies \Diamond_{[3,5]} \mathit{grant})$[1] specifies in addition that the *grant*s shall happen *within* 3 *to* 5 *time units of the occurrence of each req*. This type of *bounded-response* property arises naturally when considering safety-critical systems such as a car's braking system, or a power plant's shutdown mechanism.

MTL formulas are usually interpreted over *dense time*, which is typically modelled using the non-negative real numbers $\mathbb{R}_{\geq 0}$.[2] Furthermore, an important

[1] The $\Box$ operator is here implicitly annotated with the interval $[0, \infty)$.

[2] By contrast, in *discrete-time* settings the underlying model is typically the non-negative integers, yielding more tractable theories that however correspond less closely to physical reality [13, 3].

distinction among real-time models is whether one assumes that the system under consideration is observed at every instant in time, leading to an *interval-based semantics* [4, 21, 14], or whether one only records a countable sequence of snapshots of the system, leading to a *point-based semantics* [11, 6, 7, 12, 13, 24]. The interval-based semantics is somewhat more intuitive if one interprets atomic MTL formulas as *state propositions*, whereas the point-based semantics lends itself more naturally to the interpretation of atomic MTL formulas as instantaneous *events* or *actions*. Our main (undecidability) result concerns the point-based semantics, and accordingly that is the semantics we focus on in this paper. As it turns out, the corresponding undecidability result in the interval-based setting has been known for quite some time; see, e.g., [11, 5].

As is the case for LTL, it is possible to extend MTL with *past* temporal operators, although this variant is seldom seen in the literature. MTL with past operators, in turn, is subsumed by a certain monadic logic of timed state sequences introduced in [6]. The satisfiability problem for this logic is shown in that paper to be undecidable. The idea is to encode the halting computations of a given Turing machine as a set of timed words, which can themselves be captured by a monadic formula: configurations of the machine can be encoded within a single unit-duration time interval, since the density of time can accommodate arbitrarily large tape contents. The formula need only specify that the configurations are accurately propagated from one time interval to the next. As a result, the formula is satisfiable iff the Turing machine has a halting computation.

This construction easily carries over to MTL *with past operators*, and in fact the key ingredient required is merely *punctuality*: the ability to specify that two events occur exactly one time unit apart from each other. Unfortunately, a small oversight led to the claim, subsequently reproduced many times—see, e.g., [5, 6, 12, 15], among others—that any logic strong enough to express forward punctuality, i.e., formulas of the form $\Box(p \implies \Diamond_{[1,1]} q)$, is automatically undecidable.

In [19] we showed this claim to be erroneous by proving that the satisfiability problem for MTL over finite timed words is decidable. Recently, we also showed that the *safety* fragment of MTL, in which all 'eventuality' operators are time-bounded, is also decidable [20]. Another important decidability result appears in [4, 21, 14], where the fragment of MTL that disallows singular intervals is proved to be decidable. Yet another decidability result for a fragment of MTL can be found in [13], exploiting digitization techniques.

In light of these developments, the question of the decidability of (standard) MTL, incorrectly considered settled for many years, took on a new urgency. This paper closes the gap by showing that the (infinite) satisfiability and model-checking problems for MTL are undecidable. The proof proceeds by establishing a strong connection between MTL formulas and a particular class of faulty Turing machines, namely *insertion channel machines with emptiness-testing*, or ICMET. Using this connection, satisfiability questions about MTL formulas can be translated back and forth to 'recurrent-state problems' for ICMETs. We show the latter to be undecidable in general from first principles.

Our undecidability result also ties up a couple of loose ends. In [19], for instance, we show that MTL formulas can be encoded into one-clock timed alternating automata (see also [18]) with a weak parity acceptance condition. MTL satisfiability then corresponds to the non-emptiness problem for these automata, which this paper therefore shows to be undecidable. Our present results also imply the undecidability of universality for one-clock Büchi timed automata [2, 17], since these can easily capture (timed encodings of) the non-recurrent/invalid computations of ICMETs, following an idea similar to that used in [3].

2 Faulty Turing Machines

A *channel machine* [1, 9, 23] consists of a finite-state automaton acting on a finite number of unbounded fifo channels (queues, buffers). We are interested in a particular type of channel machines, which we call *insertion channel machines with emptiness-testing*, or *ICMET*. An ICMET is a tuple $\mathcal{C} = (S, \mathit{init}, M, C, \Delta)$, where S is a finite set of *control states*, $\mathit{init} \in S$ is the *initial control state*, M is a finite set of *messages*, C is a finite set of *channels*, and $\Delta \subseteq S \times L \times S$ is the transition relation over label set $L = \{c!m, c?m, c{=}\emptyset \mid c \in C \wedge m \in M\}$.

Intuitively, a $c!m$-transition corresponds to writing m to the tail of channel c, a $c?m$-transition corresponds to reading m from the head of channel c, whereas a $c{=}\emptyset$-transition is only enabled if channel c is empty. The latter transitions, which we call *emptiness-testing*, are useful in the presence of *insertion errors*, as we explain shortly.

A *global state* of an ICMET $\mathcal{C}$ is a tuple $(s, \overline{x})$, where $s \in S$ is the control state and $\overline{x} \in (M^*)^C$ represents the contents of all the channels. We write x_c to denote the contents of a given channel c. The rules in Δ induce an L-labelled transition relation on the set of global states as follows: $(s, c!m, t) \in \Delta$ yields a transition $(s, \ldots, x_c, \ldots) \xrightarrow{c!m} (t, \ldots, x_c{\cdot}m, \ldots)$ that writes $m \in M$ to the tail of channel c, and leaves all other channels unchanged. Likewise, $(s, c?m, t) \in \Delta$ yields a transition $(s, \ldots, m{\cdot}x_c, \ldots) \xrightarrow{c?m} (t, \ldots, x_c, \ldots)$ that reads $m \in M$ from the head of channel c, and again leaves all other channels unchanged. Finally, $(s, c{=}\emptyset, t) \in \Delta$ yields a transition $(s, \overline{x}) \xrightarrow{c=\emptyset} (t, \overline{x})$, provided that channel c is empty, i.e., $x_c = \varepsilon$.

If the above transitions were the only ones allowed, then $\mathcal{C}$ would be an *error-free* channel machine. An ICMET, however, may suffer from insertion errors, which are represented by certain additional transitions.

Given $x, y \in M^*$, write $x \sqsubseteq y$ if x can be obtained from y by deleting any number of letters. For example, $\mathsf{HIGMAN} \sqsubseteq \underline{\mathsf{HIG}}\mathsf{H}\underline{\mathsf{M}}\mathsf{OUNT}\underline{\mathsf{A}}\mathsf{I}\underline{\mathsf{N}}$, as indicated by the underlining. Extend this relation to $(M^*)^C$ by writing $\overline{x} \sqsubseteq \overline{y}$ if, for all $c \in C$, $x_c \sqsubseteq y_c$.

Insertion errors are then introduced by extending the transition relation on global states with the following clause: if $(s, \overline{x}) \xrightarrow{\alpha} (t, \overline{y})$, $\overline{x}' \sqsubseteq \overline{x}$, and $\overline{y} \sqsubseteq \overline{y}'$, then $(s, \overline{x}') \xrightarrow{\alpha} (t, \overline{y}')$.

A *computation* of $\mathcal{C}$ is a finite or infinite sequence of transitions between global states $(s_0, \overline{x_0}) \xrightarrow{\alpha_0} (s_1, \overline{x_1}) \xrightarrow{\alpha_1} (s_2, \overline{x_2}) \xrightarrow{\alpha_2} \cdots$, with $s_0 = \mathit{init}$.[3]

Note 1. Channel machines with insertion errors were first considered in [9], and later used to obtain complexity lower bounds for real-time verification problems in [19, 2]. Emptiness-testing is a well-known computational device (used, for example, in counter machines), which however adds no intrinsic power to *error-free* channel machines. In the presence of insertion errors, emptiness-testing provides a restricted amount of error detection, yet this combination has, to the best of our knowledge, never been studied before. As Theorem 2 indicates, the resulting class ICMET has computational power strictly between that of channel machines with insertion errors and that of perfect channel machines, and turns out to be a useful tool to study Metric Temporal Logic.

We are interested in the following decision problems concerning ICMETs. Let $\mathcal{C} = (S, \mathit{init}, M, C, \Delta)$ be an arbitrary ICMET, and let $t \in S$ be a particular control state of $\mathcal{C}$. The *halting problem* (also known as the *control-state reachability problem*) asks whether there is a computation of $\mathcal{C}$ that reaches t (irrespective of the contents of the channels). The *recurrent-state problem*, on the other hand, asks whether $\mathcal{C}$ has an infinite computation that visits t infinitely often (again, irrespective of channel contents).

Theorem 2. *The halting problem for ICMETs is decidable, with non-primitive recursive complexity. The recurrent-state problem for ICMETs is undecidable.*

Note that both problems are undecidable for error-free channel machines, since these are Turing-powerful. On the other hand, both problems are trivially decidable (with polynomial-time complexity) for channel machines with insertion errors (but without emptiness-testing), since insertion errors make the contents of channels irrelevant (all read- and write-transitions of every control state are at all times enabled)—see [9]. We conclude that *emptiness-testing* imparts a genuine amount of computational power to channel machines with insertion errors, which however falls short of that of perfect channel machines.

Proof. The proof of decidability relies on the theory of well-structured transition systems [10], whereas the complexity lower bound is a corollary of Proposition 25 of [19], which itself makes use of a result of Schnoebelen [23]. Both proofs are omitted here for reasons of space.

For the purposes of this paper the most important result is the undecidability of the recurrent-state problem, and accordingly we now present the proof in detail.

Let $\mathcal{C} = (S, \mathit{init}, M, C, \Delta)$ be an ICMET. Let $m \in M$ be a message and $c \in C$ be a channel. We would first like to define a 'macro' operation, called *occurrence-testing*, that succeeds only if c does not comprise any occurrence of m.

[3] One might in addition require that $\overline{x_0} = (\varepsilon, \ldots, \varepsilon)$, but in the presence of insertion errors this constraint is pointless.

To this end, assume that $\mathcal{C}$ has an extra working channel, called *temp*. To perform occurrence-testing for message m on channel c, do the following:

1. Repeatedly read off messages from c and copy them onto *temp*; if any of these messages turn out to be m, halt.
2. At some point, nondeterministically do an emptiness test on c, i.e., proceed if c is empty, otherwise halt. This guarantees that the whole of c has been copied onto *temp*.
3. Copy back the contents of *temp* onto c, ascertaining success by doing an emptiness test on *temp*.

Bearing in mind that insertion errors can occur at any time, the only conclusions that can be drawn from a successful '$m \notin c$'-occurrence-test are that (i) immediately prior to performing occurrence-testing, m did not occur within c, and (ii) upon completing occurrence-testing, c comprises at least all of its original contents, in the right order.

In what follows, occurrence-testing will repeatedly be invoked as if it were a *bona fide* atomic operation. In fact, we will also perform occurrence-testing for sets of messages, to be understood as a sequence of occurrence-tests for each element.

Let $\mathcal{T}$ be a deterministic one-tape Turing machine with tape alphabet Σ. Assume that in any infinite computation of $\mathcal{T}$ the tape contents grows unboundedly. (If this is not outright the case, simply augment $\mathcal{T}$ with a counter that periodically gets incremented.) For technical reasons, assume also that once the tape head visits a particular cell, that cell is never blank afterwards (a blank cell is represented by the symbol $B \in \Sigma$; the assumption is therefore that B can only be read by the head, but not written). The (suitably defined) halting problem for $\mathcal{T}$, when starting on a blank tape, is well-known to be undecidable.

It is equally well-known that a Turing machine such as $\mathcal{T}$ can easily be simulated by an *error-free* channel machine [8]. A single channel is required, which is used to mimic the tape of $\mathcal{T}$. The set M of channel messages includes $\{a, \widehat{a} \mid a \in \Sigma\}$. The 'hatted' versions of the symbols are used to indicate the current position of the head on the tape—accordingly, a channel should always comprise exactly one hatted symbol, except perhaps during the simulation of a head transition. M may contain other messages, to keep track, for example, of the leftmost and rightmost tape letters, etc.

The channel machine simulates a head transition by cycling through the entire channel once. Moving the head one cell to the right is straightforward, whereas the easiest way to move the head one cell to the left is to use nondeterminism: guess the new head position, and carry on with the simulation only if it is subsequently immediately confirmed that the chosen cell was the right one. All other transitions of the Turing machine are equally straightforward to simulate.

Note that nothing precludes the above procedure from being carried out by a channel machine that suffers from insertion errors; in that case, however, the 'simulation' is not guaranteed to accurately reflect the behaviour of $\mathcal{T}$.

A *space-bounded* computation of $\mathcal{T}$ is one in which $\mathcal{T}$ uses no more than some fixed, predetermined number of tape cells (say n). To simulate such a computation, one initialises the channel with exactly n blanks, and afterwards strictly alternates read-transitions with write-transitions. In other words, in the absence of insertion errors the channel size remains essentially constant having at all times either n or $n-1$ messages.[4] The simulation proceeds until the channel machine, in attempting to access a blank, is unable to do so. Note that in the presence of insertion errors, the absence of blanks can be ascertained by occurrence-testing for B.

Given $\mathcal{T}$ as above, we construct an ICMET $\mathcal{C} = (S, \mathit{init}, M, C, \Delta)$ composed of several components (cf. Figure 1). One of these components is a 'space-bounded simulator' for $\mathcal{T}$. The simulator has its own dedicated channel, which is at the beginning initialised with a certain number of blanks. (One can ascertain that only blanks are initially on the channel by occurrence-testing for every other message in M.) The simulator then simulates $\mathcal{T}$ until either $\mathcal{T}$ halts, in which case $\mathcal{C}$ also halts, or until all blanks are exhausted, in which case the simulator subroutine returns.

This space-bounded simulator is embedded within a 'decreasing device'. Once all blanks are exhausted and the simulator returns, the decreasing device does the following: it cycles through the whole channel of the simulator and re-initialises every symbol to B (ascertaining success via occurrence-testing). It then deletes one of the Bs and launches a fresh new simulation of $\mathcal{T}$ all over again.

The decreasing device only returns when, upon having re-initialised the simulator's channel with blanks and deleted one, it finds the channel to be empty.

$\mathcal{C}$ also keeps a counter, encoded in unary (using the symbol B, say), which starts at 1 and is subsequently periodically incremented. This counter is at all times stored either on channel *count* or channel *count*$'$. The role of the counter is to indicate how many blanks are to be initially provided upon freshly entering a decreasing-device cycle. This proceeds as follows. Assuming that the counter is stored on channel *count*, for every B in *count* a B is written both onto the simulator's channel and onto *count*$'$. This continues until *count* is empty, at which point control is passed to the space-bounded simulator. (The next time around proceeds similarly except that the roles of *count* and *count*$'$ are inverted, and so on.)

Once the decreasing device returns—upon finding the simulator's channel empty, as explained earlier—$\mathcal{C}$ visits a distinguished control state $t \in S$, increments the counter, and starts a new cycle.

The ICMET $\mathcal{C}$ is represented diagrammatically in Figure 1. Note that, while $\mathcal{T}$ is a deterministic Turing machine, $\mathcal{C}$ is a nondeterministic channel machine.

We claim that $\mathcal{C}$ has an infinite computation that visits control state t infinitely often iff $\mathcal{T}$ does not halt when started on a blank tape. It immediately follows that the recurrent-state problem for ICMETs is undecidable.

[4] Note that in the 'unconstrained' simulation of $\mathcal{T}$ described earlier, the channel may grow unboundedly (even without insertion errors) as the channel machine periodically adds blanks to it as needed to accurately mimic $\mathcal{T}$'s *unbounded* tape.

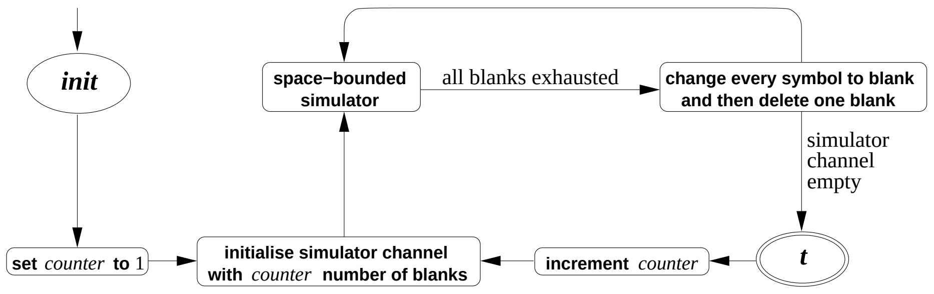
init
space−bounded simulator
all blanks exhausted
change every symbol to blank and then delete one blank
simulator channel empty
t
set counter to 1
initialise simulator channel with counter number of blanks
increment counter

Fig. 1. A schematic representation of the ICMET $\mathcal{C}$. The starting state is *init*, and we are interested in computations that visit state t infinitely often. Note that $\mathcal{C}$ is a nondeterministic machine; the two transitions emerging from 'change every symbol to blank and then delete one blank', for instance, are not mutually exclusive.

It remains to establish the claim. The right-to-left implication is immediate: if $\mathcal{T}$ does not halt, then consider an error-free computation of $\mathcal{C}$. Since by assumption $\mathcal{T}$'s tape contents grows unboundedly with time, every space-bounded simulation of $\mathcal{T}$ eventually exhausts all blanks (recall that $\mathcal{T}$ never writes a fresh blank). As the starting number of blanks in successive space-bounded simulations always decreases by one, the channel always eventually becomes empty and $\mathcal{C}$ therefore always eventually reaches t.

Assume now that $\mathcal{C}$ has an infinite computation that visits t infinitely often, and suppose on the contrary that $\mathcal{T}$ halts. Let n be the total number of tape cells visited by $\mathcal{T}$ in the course of its halting computation. Since $\mathcal{C}$ always increments its counter after visiting t, and since insertion errors can only increase, but not decrease, the value of the counter, eventually the counter reaches some value greater than or equal to n.

At that point, $\mathcal{C}$ initiates a space-bounded simulation of $\mathcal{T}$ starting with p_1 blanks, where $p_1 \geq n$. The simulation continues until no blanks remain on the tape, at which point all symbols are converted to blanks and one blank is deleted. Let us say there are then p_2 blanks on the channel. Note that, although $\mathcal{C}$ never 'knowingly' inserts any extra symbol (blank or otherwise) on the simulator's channel during a space-bounded simulation, insertion errors can occur, so that p_2 could be larger than p_1. In fact, it is clear that $p_2 = p_1 - 1$ iff no insertion error occurred throughout the entire space-bounded simulation (including the channel re-initialisation step).

Continuing in this way, we get a sequence of numbers $p_1, p_2, p_3, \ldots, p_k$ which denote the number of blanks on the channel at the beginning of every space-bounded simulation. Since by assumption the computation of $\mathcal{C}$ we are considering always eventually visits t, and since t can only be reached if the simulator channel is empty, we have $p_k = 0$. Since $p_1 \geq n$, and since the number of blanks decreases by at most 1 in going from any p_i to p_{i+1}, we conclude that there is some j such that $p_j = n$ and $p_{j+1} = n - 1$. In other words, the j-th space-bounded simulation was an *error-free* simulation of $\mathcal{T}$ that started on a channel with n blanks. Since $\mathcal{T}$ is deterministic, this simulation should have led to $\mathcal{T}$ halting, which in turn should have halted $\mathcal{C}$ as well, contradicting our initial hypothesis.

This concludes the proof of Theorem 2. □

3 Metric Temporal Logic

We now formally present Metric Temporal Logic (MTL). Given the leisurely background review offered in the Introduction, the present treatment is rather succinct. For a more detailed and comprehensive account of MTL we refer the reader to [6].

A *time sequence* $\tau = \tau_0\tau_1\ldots$ is a finite or infinite sequence of time values $\tau_i \in \mathbb{R}_{\geq 0}$ with $\tau_i \leq \tau_{i+1}$ for all $i < |\tau| - 1$. Here $|\tau|$ denotes the length of τ. If τ is infinite, we require that $\{\tau_i \mid i \in \mathbb{N}\}$ be unbounded (non-Zenoness).

A *timed word* over finite alphabet Σ is a pair $\rho = (\sigma, \tau)$, where $\sigma = \sigma_0\sigma_1\ldots$ is a word over Σ and τ is a time sequence of the same length. We also occasionally refer to a pair (σ_i, τ_i) as a *timed event*, having τ_i as a *timestamp*. Finally, we write $T\Sigma^*$ for the set of finite timed words over alphabet Σ, and $T\Sigma^\omega$ for the set of infinite timed words over Σ.[5]

Given a finite alphabet Σ of atomic events, the formulas of MTL are built up from Σ by Boolean connectives and time-constrained versions of the temporal operators *next* ($\bigcirc$), *eventually* ($\Diamond$), *always* ($\Box$), and *until* ($\mathcal{U}$), as follows:

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

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

Given a (finite or infinite) timed word $\rho = (\sigma, \tau)$ and an MTL formula φ, the satisfaction relation $(\rho, i) \models \varphi$ (read ρ satisfies φ at position i) is defined inductively as follows:

- $(\rho, i) \models \top$
- $(\rho, i) \models \varphi_1 \wedge \varphi_2$ iff $(\rho, i) \models \varphi_1$ and $(\rho, i) \models \varphi_2$
- $(\rho, i) \models \neg\varphi$ iff $(\rho, i) \not\models \varphi$
- $(\rho, i) \models a$ iff $i < |\rho|$ and $\sigma_i = a$
- $(\rho, i) \models \bigcirc_I \varphi$ iff $i + 1 < |\rho|$, $(\rho, i+1) \models \varphi$, and $\tau_{i+1} - \tau_i \in I$
- $(\rho, i) \models \Diamond_I \varphi$ iff there exists j such that $i \leq j < |\rho|$, $(\rho, j) \models \varphi$, and $\tau_j - \tau_i \in I$
- $(\rho, i) \models \Box_I \varphi$ iff for all j such that $i \leq j < |\rho|$, if $\tau_j - \tau_i \in I$ then $(\rho, j) \models \varphi$

[5] Note that we are adopting a *weakly monotonic* view of time, in that several events are allowed to share the same timestamp. The results presented here however carry over verbatim under a strongly monotonic interpretation of time.

– $(\rho, i) \models \varphi_1 \, \mathcal{U}_I \, \varphi_2$ iff there exists j such that $i \leq j < |\rho|$, $(\rho, j) \models \varphi_2$, $\tau_j - \tau_i \in I$, and $(\rho, k) \models \varphi_1$ for all k with $i \leq k < j$.

We say that ρ satisfies φ, denoted $\rho \models \varphi$, if $(\rho, 0) \models \varphi$. Additional Boolean and temporal operators can be defined via the usual conventions. Note that the expected relationships among the constrained temporal operators hold, viz. $\Diamond_I \varphi \equiv \top \, \mathcal{U}_I \, \varphi$ and $\Box_I \varphi \equiv \neg \Diamond_I \neg \varphi$. We have nonetheless defined $\Diamond_I$ and $\Box_I$ separately because our main undecidability result does not require the $\mathcal{U}_I$ operators.

Büchi timed automata [3] are real-time extensions of Büchi automata that accept infinite timed words. It is not necessary for our purposes to say anything more about these, other than to state that there exists a (rather trivial) Büchi timed automaton that accepts $T\Sigma^\omega$, the set of all infinite timed words.

Given an MTL formula φ, the *finite-satisfiability problem* asks if there exists a finite timed word that satisfies φ; this problem was shown to be decidable, with non-primitive recursive complexity, in [19]. The *infinite-satisfiability problem* asks if there is an infinite timed word that satisfies φ. Finally, the *infinite model-checking problem* asks, given a Büchi timed automaton $\mathcal{A}$, whether all infinite timed words accepted by $\mathcal{A}$ satisfy φ. The main result of this paper is the following:

Theorem 3. *The infinite-satisfiability and infinite model-checking problems for MTL are undecidable. In fact, these problems are already undecidable for the fragment of MTL that excludes all constrained 'until' operators $\mathcal{U}_I$.*

Proof. The infinite-satisfiability part follows immediately from Theorem 4.3 (in the next section) and Theorem 2.

For the infinite model-checking statement, consider a universal Büchi timed automaton (i.e., one that accepts every timed word). Model checking this automaton against an MTL formula is equivalent to asking whether the formula is valid, i.e., whether its negation is unsatisfiable. □

4 Two-way Reductions

We exhibit a correspondence between the faulty Turing machines studied in Section 2 and Metric Temporal Logic formulas. More precisely, we show how to effectively translate finite (respectively infinite) MTL satisfiability questions into halting (respectively recurrent-state) problems for ICMETs, and vice-versa.

The advantage of this correspondence is that many questions about MTL, whose dense-time semantics is sometimes considered somewhat awkward and counter-intuitive [15], can be translated into the purely discrete framework of ICMETs.

Theorem 4. *The following reductions between ICMETs and MTL formulas are all effective:*

1. *Let $(\mathcal{C}, t)$ be an instance of the halting problem for ICMETs. Then there exists an MTL formula φ such that $\mathcal{C}$ reaches t iff φ is satisfiable by some finite word.*

2. *Let φ be an MTL formula. Then there exists an ICMET $\mathcal{C}$ together with a distinguished control state t of $\mathcal{C}$ such that φ is satisfiable by some finite word iff $\mathcal{C}$ reaches t.*
3. *Let $(\mathcal{C}, t)$ be an instance of the recurrent-state problem for ICMETs. Then there exists an MTL formula φ such that $\mathcal{C}$ has a t-recurrent computation iff φ is satisfiable by some infinite word.*
4. *Let φ be an MTL formula. Then there exists an ICMET $\mathcal{C}$ together with a distinguished control state t of $\mathcal{C}$ such that φ is satisfiable by some infinite word iff $\mathcal{C}$ has a t-recurrent computation.*

Moreover, for statements 1 and 3 the fragment of MTL that excludes all constrained 'until' operators $\mathcal{U}_I$ suffices.

Proof. For the purposes of this paper the most important statement is 3, and accordingly we give full details of that proof and briefly comment on the other cases afterwards.

Let $\mathcal{C} = (S, \mathit{init}, M, C, \Delta)$ be an ICMET, with $t \in S$ the distinguished control state. The idea is to encode valid t-recurrent computations of $\mathcal{C}$ as timed words, that are in turn captured by an MTL formula φ.

To this end, assume that $\mathcal{C}$ has k channels, say $C = \{c_1, \ldots, c_k\}$. Define an alphabet $\Sigma = S \cup M \cup \Delta \cup \{B_i, E_i \mid 1 \leq i \leq k\}$. A global state $(s, x_1, \ldots, x_k)$ of $\mathcal{C}$ is encoded as a finite timed word of total duration $2k$, as follows:

- s occurs at time 0.
- The contents x_i of channel c_i is encoded in the open interval $(2i-1, 2i)$ as a matching sequence of timed events. The latest event corresponds to the message at the head of the channel, and so on.
- The event B_i occurs at time $2i-1$, and the event E_i occurs at time $2i$. These two events therefore delineate the contents of channel c_i.

Moreover, the timed word is strongly monotonic (no two timed events share the same timestamp), and contains no timed events other than the ones listed above. In particular, the open time intervals $(2i, 2i+1)$ are empty.

Note that the density of time allows such timed words to accommodate arbitrarily large channel contents. Note also that any such timed word can be uniquely converted into a global state of $\mathcal{C}$.

A computation $(s_0, \overline{x_0}) \xrightarrow{\alpha_0} (s_1, \overline{x_1}) \xrightarrow{\alpha_1} (s_2, \overline{x_2}) \xrightarrow{\alpha_2} \cdots$ of $\mathcal{C}$ can then be encoded as an infinite timed word, by time-shifting and concatenating the timed words corresponding to each global state and interspersing the transitions α_j, as follow:

- If s_0 occurs at time τ_0, then s_j occurs at time $(2k+2)j + \tau_0$, followed by the encoding of the remainder of the j-th global state, as detailed above.
- α_j occurs at time $(2k+2)(j+1) - 1 + \tau_0$, i.e., exactly one time unit after the end of the encoding of the j-th global state, and exactly one time unit before event s_{j+1}.

– If message m in channel c_i, in global state j, is not read off while performing the transition α_j, then the difference between the timestamps of the two occurrences of m in the encodings of the $(j{+}1)$-th and j-th global states is exactly $2k+2$.

The last clause ensures that channel contents are preserved between transitions; nothing prevents, however, insertion errors from occurring, in the form of timed events with no matching events $2k+2$ time units *earlier*.

Observe that any infinite computation of $\mathcal{C}$ can immediately be recovered from its encoding as an infinite timed word.

It remains to exhibit an MTL formula φ that captures precisely the timed words corresponding to the t-recurrent infinite computations of $\mathcal{C}$. We first build various useful components, as follows:

We first want to restrict ourselves to strongly monotonic timed words:

$$\varphi_{\mathsf{sm}} = \Box \bigcirc_{>0} \top .$$

The first event is the control state *init*, and afterwards control states are forever spaced exactly $2k+2$ time units apart:

$$\varphi_S = \mathit{init} \wedge \Box \left(\bigvee S \implies \left(\Diamond_{=2k+2} \bigvee S \wedge \Box_{<2k+2} \neg \bigvee S \right) \right) .$$

The structure of global-state encodings is captured by the following formulas, for $1 \le i \le k$:

$$\begin{aligned}
\varphi_{B_i} &= \Box \left(\bigvee S \implies \left(\Diamond_{=2i-1} B_i \wedge \Box_{[0,2i-1)\cup(2i-1,2k+2)} \neg B_i \right) \right) \\
\varphi_{E_i} &= \Box \left(\bigvee S \implies \left(\Diamond_{=2i} E_i \wedge \Box_{[0,2i)\cup(2i,2k+2)} \neg E_i \right) \right) \\
\varphi_{c_i} &= \Box \left(\bigvee S \implies \left(\Box_{(2i-1,2i)} \bigvee M \wedge \Box_{(2i,2i+1)} \bot \right) \right) .
\end{aligned}$$

Interspersing transitions:

$$\varphi_{\Delta} = \Box \left(\bigvee S \implies \left(\Diamond_{=2k+1} \bigvee \Delta \wedge \Box_{<2k+1} \neg \bigvee \Delta \wedge \Box_{(2k+1,2k+2)} \bot \right) \right) .$$

We now define components that ensure the validity of the encoded computation.

Consecutive control states should match the source and target of the intervening transitions; to this end, for any pair of control states s, s', let $\Delta_{s,s'} = \{(s, -, s') \in \Delta\}$.

$$\varphi_{\Delta S} = \bigwedge_{s,s' \in S} \Box \left((s \wedge \Diamond_{=2k+2} s') \implies \Diamond_{=2k+1} \bigvee \Delta_{s,s'} \right) .$$

To handle channel integrity, first define:

$$\varphi_{\mathsf{copy}} = \Box_{(0,1)} \bigwedge_{m \in M} (m \implies \Diamond_{=2k+2} m) .$$

Then, for $1 \leq i \leq k$ and $m \in M$, let:

$$\varphi_{c_i=\emptyset} = \bigvee S \wedge \bigwedge_{1\leq j\leq k} \Diamond_{=2j-1}\varphi_{\mathsf{copy}} \wedge \Box_{(2i-1,2i)}\bot$$

$$\varphi_{c_i!m} = \bigvee S \wedge \bigwedge_{1\leq j\leq k} \Diamond_{=2j-1}\varphi_{\mathsf{copy}} \wedge \Diamond_{[2i-1,2i)}\left(\bigcirc E_i \wedge \Diamond_{=2k+2}\bigcirc m\right)$$

$$\varphi_{c_i?m} = \bigvee S \wedge \bigwedge_{\substack{1\leq j\leq k\\ j\neq i}} \Diamond_{=2j-1}\varphi_{\mathsf{copy}} \wedge \Diamond_{=2i-1}\bigcirc\left(m \wedge \varphi_{\mathsf{copy}}\right).$$

Channel contents should vary according to the relevant transitions. Recall that $L = \{c!m, c?m, c{=}\emptyset \mid c \in C \wedge m \in M\}$. For $l \in L$, let $\Delta_l = \{(-, l, -) \in \Delta\}$.

$$\varphi_{\Delta C} = \Box\left(\bigvee S \implies \bigwedge_{l\in L}\left(\Diamond_{=2k+1}\bigvee \Delta_l \implies \varphi_l\right)\right),$$

where the formulas φ_l are defined above.

We are interested in t-recurrent computations of $\mathcal{C}$, which are captured by requiring:

$$\varphi_{\mathsf{rec}} = \Box\Diamond t.$$

Finally, let:

$$\varphi = \varphi_{\mathsf{sm}} \wedge \varphi_S \wedge \bigwedge_{1\leq i\leq k}\left(\varphi_{B_i} \wedge \varphi_{E_i} \wedge \varphi_{c_i}\right) \wedge \varphi_\Delta \wedge \varphi_{\Delta S} \wedge \varphi_{\Delta C} \wedge \varphi_{\mathsf{rec}}.$$

By construction, infinite timed words that satisfy φ can be translated into valid t-recurrent computations of $\mathcal{C}$, and vice-versa. It is also clear that φ does not use any $\mathcal{U}_I$ operator, concluding the proof of Statement 3.

Note that a proof of Statement 1 can easily be engineered along the same lines as the above.

For Statement 4, one first reduces infinite satisfiability for MTL to a non-emptiness problem for one-clock timed alternating automata with a weak parity acceptance condition, by extending the construction presented in [19]. Next, one translates this non-emptiness problem into the existence of a Büchi path in a certain well-structured transition system, which can itself be described using a perfect channel machine, again following a construction of [19]. One then argues that insertion errors can only cause valid Büchi paths to be rejected, thereby preserving correctness.

Finally, Statement 2 can be handled by following a simplified version of the above procedure. □

5 Summary

The main result of this paper is that the satisfiability and model checking problems for Metric Temporal Logic, interpreted over infinite timed words, are undecidable. As such, this closes a gap between a host of decidability and undecidability results for various variants of MTL. The crux of our approach is to establish a strong correspondence between problems about Metric Temporal Logic and problems about ICMETs, a particular brand of faulty Turing machines, as depicted in Figure 2.

MTL	**ICMET**	Complexity
Finite satisfiability	Halting problem	Non-primitive recursive
Infinite satisfiability	Recurrent-state problem	Undecidable

Fig. 2. A summary of the two-way reductions between Metric Temporal Logic and faulty Turing machine problems.

An interesting question is whether this correspondence can be leveraged, in one direction or the other, to obtain additional results or insights about the two entities MTL and ICMET.

References

[1] P. Abdulla and B. Jonsson. Undecidable verification problems with unreliable channels. *Inf. Comput.*, 130:71–90, 1996.

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

[3] R. Alur and D. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, 1994.

[4] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43:116–146, 1996.

[5] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *Proc. RTTP*, volume 600 of *Springer LNCS*, 1992.

[6] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104:35–77, 1993.

[7] R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41:181–204, 1994.

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

[9] G. Cécé, A. Finkel, and S. P. Iyer. Unreliable channels are easier to verify than perfect channels. *Inf. Comput.*, 124:20–31, 1996.

[10] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

[11] T. A. Henzinger. *The Temporal Specification and Verification of Real-Time Systems.* PhD thesis, Stanford University, 1991. Tech. rep. STAN-CS-91-1380.

[12] T. A. Henzinger. It's about time: Real-time logics reviewed. In *Proc. CONCUR*, volume 1466 of *Springer LNCS*, 1998.

[13] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proc. ICALP*, volume 623 of *Springer LNCS*, 1992.

[14] T. A. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In *Proc. ICALP*, volume 1443 of *Springer LNCS*, 1998.

[15] Y. Hirshfeld and A. Rabinovich. Logics for real time: Decidability and complexity. *Fundam. Inform.*, 62(1):1–28, 2004.
[16] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time Systems*, 2(4):255–299, 1990.
[17] S. Lasota and I. Walukiewicz. Personal communication, 2005.
[18] S. Lasota and I. Walukiewicz. Alternating timed automata. In *Proc. FOSSACS*, volume 3441 of *Springer LNCS*, 2005.
[19] J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *Proc. LICS*. IEEE Press, 2005.
[20] J. Ouaknine and J. Worrell. Safety metric temporal logic is fully decidable. Submitted, 2005.
[21] J.-F. Raskin and P.-Y. Schobbens. State-clock logic: A decidable real-time logic. In *Proc. HART*, volume 1201 of *Springer LNCS*, 1997.
[22] K. Schneider. *Verification of Reactive Systems.* Springer, 1997.
[23] P. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Inf. Process. Lett.*, 83(5):251–261, 2002.
[24] T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Proc. FTRTFTS*, volume 863 of *Springer LNCS*, 1994.