Information and Computation 298 (2024) 105168

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/yinco

Model checking timed recursive CTL

Florian Bruse *, Martin Lange *

Theoretical Computer Science / Formal Methods, University of Kassel, Germany

ARTICLE INFO

Article history:
Received 28 June 2022
Received in revised form 22 March 2024
Accepted 4 April 2024
Available online 9 April 2024

Keywords:
Timed automata
Model checking

ABSTRACT

We introduce Timed Recursive CTL, a merger of two extensions of the well-known branching-time logic CTL: Timed CTL is interpreted over real-time systems like timed automata; Recursive CTL introduces a powerful recursion operator which takes the expressiveness of this logic CTL well beyond that of regular properties. The result is an expressive logic for real-time properties. We show that its model checking problem is decidable over timed automata, namely 2-EXPTIME-complete.

1. Introduction

Temporal logics are widely used as formal languages for the specification of properties of reactive systems. The most commonly known such logics are LTL [1], CTL [2] and CTL* [3], having achieved this status – especially when it comes to LTL and CTL– partially because of their intuitive syntax. Both LTL and CTL can be seen as extensions of propositional logic by a small set of intuitive temporal operators. This simplicity in syntax is also reflected by relatively low expressive power; both LTL and CTL do not even reach up to full regularity in the sense that they are not equi-expressive to finite-state word and tree automata, respectively. In addition, CTL and LTL's incomparability in expressive power had led to discussions and studies on what is "the right" temporal logic for program specification, as well as for example the introduction of the aforementioned CTL*, unifying both of them. By now, it is clear that there is no single right temporal logic. Instead, it is the demands on expressive power and pragmatics raised by particular applications which determine what the best temporal logic for those specific purposes is.

There is, however, a common understanding of the limitations put onto a logic's usability given by its expressive power. In particular, so-called "regular" temporal logics – i.e. those that do not exceed the expressive power of corresponding finite automata models – typically have appealing computational properties like decidability of their model and satisfiability checking problems [4], finite or tree model properties [5], etc. In this way, regular expressive power is a cornerstone in the study of the theory of temporal specification languages, and when exceeding it one should expect to lose some of these properties. On the other hand, one also gains expressive power by definition when extending the expressive power of a logic beyond regularity, and there are interesting program properties which are not regular and can therefore not be expressed in such logics, like the absence of buffer over-/underflows [6], assume-guarantee properties [7], etc. The literature contains several non-regular extensions of temporal logics or related modal fixpoint logics, e.g. PDL[CFL] [8], FLC [9] and HFL [7]. These have certain features in common: a syntax that makes it difficult to understand the meaning of formulas, and – despite undecidability of their satisfiability problems – a decidable model checking problem over finite structures [10–12]. The upshot to take from this is that model checking need not become undecidable when going beyond regular expressiveness.

* Corresponding authors.
E-mail addresses: florian.bruse@uni-kassel.de (F. Bruse), martin.lange@uni-kassel.de (M. Lange).

https://doi.org/10.1016/j.ic.2024.105168

In order to overcome issues with unintuitive syntaxes in expressive specification languages, a temporal logic called Recursive CTL (RecCTL) was recently proposed [13]. It extends the basic branching-time temporal logic CTL with a single recursion operator which takes formulas as arguments that can be manipulated using other temporal and Boolean operators and then be passed into a recursive call. This achieves expressive power, capturing all regular branching-time properties and many non-regular ones. The former is due to the fact that, semantically, recursion is explained via least fixpoints (as it is common in programming language semantics). So whilst syntactically, RecCTL extends the fairly simple CTL, semantically it is rather an extension of the modal μ-calculus [14], the archetypical yet unintuitive regular program logic [15].

As mentioned above, decidability of model checking for such logics can be retained, but at the cost of higher computational complexity. For RecCTL, it is exponentially worse than for CTL, being EXPTIME-complete compared to P-completeness for CTL.

Another way of extending the expressive power of temporal logics, which has been followed in the literature for quite some time, is more semantic in nature: in the labelled transition systems that logics like CTL are interpreted over, the evolution of time is modelled abstractly via discrete steps that are taken when passing from one state to another. Hence, the only real timing properties expressible in such logics are unitless and non-quantitative like "*at some point in the future*" etc., or bound to fixed steps if discrete transitions are assigned a concrete amount of time passed. This is not sufficient for the modelling of embedded or real-time systems. For example, in [16], concrete timing constraints play a role in correctness properties, for instance as in "*within 5 milliseconds of receiving a signal, a control command is issued*."

In order to capture such effects, transition systems have been extended to model the flow of time more realistically with non-negative, real-numbered delays between time points. Timed automata [17] are a popular model for the finite representation of such systems. Their greater expressiveness compared to ordinary discrete systems is indicated by the fact that the basis for algorithmic solutions to temporal logic decision problems, the reachability problem, is already PSPACE-complete [17].

One of the most popular temporal logics for expressing more complex reachability properties of timed automata is Timed CTL (TCTL) [18], an extension of CTL that is capable of making simple assertions about the amount of time that passes before certain events occur on some paths, or on all paths. Its model checking problem over timed automata is not more difficult than simple reachability: it is also PSPACE-complete [19].

Here we introduce and study Timed Recursive CTL (TRCTL), a logic that arises from combining the extensions to real-time on one hand, and to non-regular properties on the other. We show that TRCTL retains decidability of model checking over timed automata, but the combination increases the complexity to 2-EXPTIME-completeness.

The paper is organised as follows. In Sect. 2 we recall necessary preliminaries about timed automata, TCTL, and about RecCTL. In Sect. 3 we then introduce TRCTL formally. In Sect. 4 and Sect. 5 we establish 2-EXPTIME-completeness of its model checking problem. The upper bound is obtained by an exponential reduction to the RecCTL model checking problem, making use of the known region graph abstraction. This happens in Sect. 4. The lower bound, given in Sect. 5, makes use of the possibility to encode large numbers in the clock values of timed automata and TRCTL's ability to manipulate them in a way that simulates a suitable game problem. This game problem is complete for the class 2-EXPTIME and provides a suitable intermediate problem for a reduction from the generic word problem for doubly-exponential time-bounded Turing machines and model checking TRCTL. It uses the same principles that can also be found in the proof of the theorem stating that alternating $s(n)$-space bounded Turing machines can simulate deterministic $2^{\mathcal{O}(s(n))}$-time bounded Turing machines [20].

The lower bound presented here strengthens a corresponding result in a preliminary version of this paper [21], where 2-EXPTIME-hardness was established for TRCTL's model checking problem over timed automata with an unbounded number of clocks. In fact, the number of clocks used in that proof was linear. More precisely, it was shown that there is a family $(\mathcal{A}_n)_{n\geq 1}$ of timed automata with $\mathcal{A}_n$ using $n+1$ clocks such that model checking TRCTL over any class of systems containing this family is 2-EXPTIME-hard. Here we improve the lower bound and show that a single clock suffices for 2-EXPTIME-hardness already. Moreover, this holds already for the expression complexity of the TRCTL model checking problem, i.e. the hardness result is obtained over a fixed automaton, and we also show that already quite simple fragments of TRCTL admit the hardness result, in contrast to a quite diverse complexity landscape in the TCTL setting [19].

We conclude in Sect. 6 with remarks on possibilities to extend this work.

2. Preliminaries

For $n > 0$, we write $[n]$ for the set $\{0, \dots, n-1\}$.

2.1. Models of real-time systems

We recall timed automata as models of (infinite-state) real-time systems and the untiming construction known as the region-graph construction or abstraction [17] that is the basis for decidability proofs about reachability in and model checking over such systems.

2.1.1. Timed transition systems

A *timed labelled transition system* (TLTS) over a finite set *Prop* of atomic propositions is a $\mathcal{T} = (\mathcal{S}, \to, s_0, \lambda)$ such that

- $\mathcal{S}$ is a set of *states* containing a designated starting state s_0,
- $\rightarrow \subseteq \mathcal{S} \times \mathcal{S} \cup \mathcal{S} \times \mathbb{R}^{\geq 0} \times \mathcal{S}$ is the transition relation, consisting of two kinds:
  - *discrete transitions* of the form $s \rightarrow t$ for $s, t \in \mathcal{S}$, and
  - *delay transitions* of the form $s \xrightarrow{d} t$ for $s, t \in \mathcal{S}$ and $d \in \mathbb{R}^{\geq 0}$, satisfying $s \xrightarrow{0} t$ iff $s = t$ for any $s, t \in \mathcal{S}$, and

$$\forall d, d_1, d_2 \in \mathbb{R}^{\geq 0} \text{ such that } d = d_1 + d_2, \forall s, t \in \mathcal{S} : s \xrightarrow{d} t \Leftrightarrow \exists u \in \mathcal{S} \text{ such that } s \xrightarrow{d_1} u \text{ and } u \xrightarrow{d_2} t \ ,$$

- $\lambda : \mathcal{S} \rightarrow 2^{Prop}$ labels the states with the set of atomic propositions that hold true in it.

The *extended transition relations* $\xRightarrow{d}$, $d \in \mathbb{R}^{\geq 0}$, are obtained by padding discrete transitions with delays:

$$s \xRightarrow{d} t \quad \text{iff} \quad \exists d_1, d_2 \in \mathbb{R}^{\geq 0}, s', t' \in \mathcal{S} \text{ such that } s \xrightarrow{d_1} s', s' \rightarrow t', t' \xrightarrow{d_2} t \text{ and } d = d_1 + d_2$$

A *trace* is a sequence $\pi = s_0 \xRightarrow{d_0} s_1 \xRightarrow{d_1} \dots$

Note that we only consider TLTS over a singleton set of discrete actions. This is done purely since the temporal logics based on CTL used in this paper do not consider different actions. It would be possible to extend the entire theory to TLTS over several discrete transition relations $\xrightarrow{a}, \xrightarrow{b}, \dots$, and make the logics aware of these.

An (untimed) labelled transition system (LTS) is a TLTS over an empty delay transition relation. It is *finite* if the set of its states is finite.

2.1.2. Clock constraints

Let $\mathcal{X} = \{\mathrm{x}, \mathrm{y}, \dots\}$ be a set of $\mathbb{R}^{\geq 0}$-valued variables called *clocks*. By $CC(\mathcal{X})$ we denote the set of *clock constraints* over $\mathcal{X}$ which are conjunctions of formulas of the form $\top$ or $\mathrm{x} \oplus c$ for $\mathrm{x} \in \mathcal{X}$, $c \in \mathbb{N}$ and $\oplus \in \{\leq, <, \geq, >, =\}$.

A *clock evaluation* is a function $\eta : \mathcal{X} \rightarrow \mathbb{R}^{\geq 0}$. A clock constraint φ is interpreted in a clock evaluation η in the obvious way:

- $\eta \models \top$ holds for any η,
- $\eta \models \varphi_1 \wedge \varphi_2$ if $\eta \models \varphi_1$ and $\eta \models \varphi_2$,
- $\eta \models \mathrm{x} \oplus c$ if $\eta(\mathrm{x}) \oplus c$ for $\oplus \in \{\leq, <, \geq, >, =\}$.

Given a clock evaluation η, $d \in \mathbb{R}^{\geq 0}$ and a set $R \subseteq \mathcal{X}$, we write $\eta{+}d$ for the clock evaluation that is defined by $(\eta{+}d)(\mathrm{x}) = \eta(\mathrm{x}) + d$ for any $\mathrm{x} \in \mathcal{X}$, and $\eta|_R$ for the clock evaluation that is defined by $\eta|_R(\mathrm{x}) = 0$ for $\mathrm{x} \in R$ and $\eta|_R(\mathrm{x}) = \eta(\mathrm{x})$ otherwise.

2.1.3. Timed automata

As with TLTS, here we consider timed automata whose transitions are always taken with a single action which is consequently not named. As above, the reason for considering this simplified model is purely the fact that CTL-based logics - the main object of study in this paper – are oblivious of differences in actions anyway. For a detailed introduction into timed automata see e.g. [17,22].

A *timed automaton* (TA) over *Prop* is a $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda)$ where

- L is a finite set of so-called *locations* containing a designated *initial* location $\ell_0 \in L$,
- $\mathcal{X}$ is a finite set of clocks,
- $\iota : L \rightarrow CC(\mathcal{X})$ assigns a clock constraint, called *invariant*, to each location,
- $\delta \subseteq L \times CC(\mathcal{X}) \times 2^{\mathcal{X}} \times L$ is a finite set of transitions,
- $\lambda : L \rightarrow 2^{Prop}$ is a propositional labelling.

Note that we write $\ell \xrightarrow{g,R} \ell'$ instead of $(\ell, g, R, \ell') \in \delta$. In such a transition, g is called the *guard*, and $R \subseteq \mathcal{X}$ are the *reset* clocks of this transition.

The *index* of the TA $\mathcal{A}$ is the largest constant occurring in its invariants or guards, denoted $m(\mathcal{A})$. The *size* of $\mathcal{A}$ is defined as usual (see e.g. [22]) via

$$|\mathcal{A}| = |\delta| \cdot (2 \cdot (\log L) + |\mathcal{X}| + \log m(\mathcal{A})) + |L| \cdot 2 \cdot (\log |\mathcal{X}| + \log m(\mathcal{A})) + |L| \cdot |Prop|.$$

Note that the size is only logarithmic in the value of constants used in clock constraints as they can be represented in binary notation, for instance.

TA are models of state-based real-time systems. The semantics, or behaviour of a given TA $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda_{\mathcal{A}})$ is defined by a TLTS $\mathcal{T}_{\mathcal{A}}$ over the time domain $\mathbb{R}^{\geq 0}$ as follows.

- The state set is $\mathcal{S} = \{(\ell, \eta) \in L \times (\mathcal{X} \rightarrow \mathbb{R}^{\geq 0}) \mid \eta \models \iota(\ell)\}$ consisting of pairs of locations and clock evaluations that satisfy the locations's invariant.

- The initial state is $s_0 = (\ell_0, \eta_0)$ where $\eta_0(\mathrm{x}) = 0$ for all $\mathrm{x} \in \mathcal{X}$.
- Delay transitions retain the underlying location and (possibly) advance the value of clocks in a state: for any $(\ell, \eta) \in \mathcal{S}$ and $d \in \mathbb{R}^{\geq 0}$ we have $(\ell, \eta) \xrightarrow{d} (\ell, \eta+d)$ if $\eta+d \models \iota(\ell)$.
- Discrete transitions possibly change the location and reset clocks: for any $(\ell, \eta) \in \mathcal{S}$, $\ell' \in L$ and $R \subseteq \mathcal{X}$ we have $(\ell, \eta) \to (\ell', \eta|_R)$ if there is $g \in CC(\mathcal{X})$ such that $(\ell, g, R, \ell') \in \delta$ and $\eta|_R \models \iota(\ell')$.
- The propositional label of a state is inherited from the propositional label of the underlying location: $\lambda(\ell, \eta) = \lambda_{\mathcal{A}}(\ell)$.
- Clock constraints hold in a state if they hold for its clocks, i.e. $(\ell, \eta) \models \chi$ iff $\eta \models \chi$.

In other words, a TA finitely represents a TLTS. Clearly, not every TLTS is finitely representable, so only a subset is captured by TA.

2.2. Temporal logics

We recall the two most relevant temporal logics which form the basis for the definition of Timed Recursive CTL in Sect. 3: Timed CTL, the extension of pure CTL by operators to quantitatively speak about the passage of time, and Recursive CTL, the extension of CTL by a recursion operator which gives it much greater expressive power.

2.2.1. Timed computation tree logic

As before, let *Prop* be a set of atomic propositions and $\mathcal{X}$ be a set of clocks. Formulas of Timed CTL (TCTL) are given by the following grammar.

$$\varphi ::= q \mid \chi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathtt{E}(\varphi \,\mathtt{U}_J\, \varphi) \mid \mathtt{A}(\varphi \,\mathtt{U}_J\, \varphi)$$

where $q \in \mathit{Prop}$, χ is a clock constraint over $\mathcal{X}$, and J denotes an interval in $\mathbb{R}^{\geq 0}$ with integer bounds, i.e. it takes one of the forms $[n, m], (n, m), [n, m), (n, m), [n, \infty), (n, \infty)$ with $n, m \in \mathbb{N}$, $n \leq m$.

Other Boolean connectives are defined as abbreviations in the usual way: $\mathtt{tt} := q \vee \neg q$ for some $q \in \mathit{Prop}$, $\mathtt{ff} = \neg\mathtt{tt}$, $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \to \psi := \neg\varphi \vee \psi$, etc. Likewise, other familiar temporal operators can be obtained as abbreviations as well: $\mathtt{Q}\,\mathtt{F}_J\varphi := \mathtt{Q}(\mathtt{tt}\,\mathtt{U}_J\,\varphi)$ for $\mathtt{Q} \in \{\mathtt{E}, \mathtt{A}\}$, $\mathtt{Q}\,\mathtt{G}_J\varphi := \neg\overline{\mathtt{Q}}\,\mathtt{F}_J\neg\varphi$ where $\overline{\mathtt{E}} = \mathtt{A}$ and $\overline{\mathtt{A}} = \mathtt{E}$. We also write certain intervals in the form $\oplus n$ with $\oplus \in \{\leq, <, \geq, >, =\}$ when possible. For instance $\mathtt{EF}_{>2}q$ stands for $\mathtt{EF}_{(2,\infty)}q$, and $\mathtt{AG}_{\leq 5}q$ stands for $\mathtt{AG}_{[0,5]}q$. Moreover, for an interval $[c, d]$, we write $\mathrm{x} \in J$ to abbreviate $\mathrm{x} \geq c \wedge \mathrm{x} \leq d$, and likewise for open and semi-open intervals.

TCTL formulas are interpreted over $\mathbb{R}^{\geq 0}$-timed transition systems $\mathcal{T} = (\mathcal{S}, \to, s_0, \lambda)$ with clock constraints as part of the propositional labelling: $[\![\varphi]\!]^{\mathcal{T}}$ denotes the set of states in $\mathcal{T}$ in which φ holds, defined inductively as follows.

$$\begin{aligned}
[\![q]\!]^{\mathcal{T}} &:= \{s \mid q \in \lambda(s)\} \\
[\![\chi]\!]^{\mathcal{T}} &:= \{s \mid s \models \chi\} \\
[\![\varphi \wedge \psi]\!]^{\mathcal{T}} &:= [\![\varphi]\!]^{\mathcal{T}} \cap [\![\psi]\!]^{\mathcal{T}} \\
[\![\neg\varphi]\!]^{\mathcal{T}} &:= \mathcal{S} \setminus [\![\varphi]\!]^{\mathcal{T}} \\
[\![\mathtt{E}(\varphi \,\mathtt{U}_J\, \psi)]\!]^{\mathcal{T}} &:= \{s \in \mathcal{S} \mid \text{there is a trace } \pi = s, \ldots \text{ such that } \pi \models \varphi \,\mathtt{U}_J\, \psi\} \\
[\![\mathtt{A}(\varphi \,\mathtt{U}_J\, \psi)]\!]^{\mathcal{T}} &:= \{s \in \mathcal{S} \mid \text{for all traces } \pi = s, \ldots \text{ we have } \pi \models \varphi \,\mathtt{U}_J\, \psi\}
\end{aligned}$$

and for an infinite non-Zeno trace[1] $\pi = s_0 \xrightarrow{d_0} s_1 \xrightarrow{d_1} s_2 \xrightarrow{d_2} \ldots$ we have $\pi \models \varphi \,\mathtt{U}_J\, \psi$ iff

$$\begin{aligned}
&\exists i \geq 0, \exists d \in [0, d_i], \exists s' \text{ such that } s_i \xrightarrow{d} s' \text{ and } (\sum_{h=0}^{i-1} d_h) + d \in J \text{ and } s' \in [\![\psi]\!]^{\mathcal{T}} \text{ and} \\
&\qquad \forall j < i, \forall d' \in [0, d_j], \forall s' \text{ such that } s_j \xrightarrow{d'} s' \text{ we have } \mathcal{T}, s' \models \varphi \vee \psi \text{ and} \\
&\qquad \forall d' \in [0, d), \forall s' \text{ such that } s_i \xrightarrow{d'} s' \text{ we have } \mathcal{T}, s' \models \varphi \vee \psi.
\end{aligned}$$

We write $\mathcal{T}, s \models \varphi$ if $s \in [\![\varphi]\!]^{\mathcal{T}}$ for arbitrary $s \in \mathcal{S}$, and also $\mathcal{T} \models \varphi$ if $\mathcal{T}, s_0 \models \varphi$.

The disjunction in the last two clauses of the semantics of the $\mathtt{U}$-operator ensures that formulas like $\mathtt{E}(\mathrm{x} = 0 \,\mathtt{U}\, \mathrm{x} > 0)$ are not unsatisfiable, which clearly would be a modelling artefact.

The *model checking problem* for TCTL is the following: given a TA $\mathcal{A}$ with clock constraints in Ξ and a TCTL formula φ, decide whether or not $\mathcal{T}_{\mathcal{A}} \models \varphi$.

[1] Recall that a Zeno trace is one on which only a finite amount of time passes, yet infinitely many discrete transitions happen.

Proposition 2.1 *([23,19]). The model checking problem for* TCTL *is* PSPACE*-complete, even for TA over a single clock.*

2.2.2. Temporal logic with recursion

We briefly present Recursive CTL (RecCTL), the other building block besides TCTL that makes up Timed Recursive CTL, to be defined in the following section.

Let *Prop* be a set of atomic propositions. Formulas of RecCTL are obtained by addition of the recursion operator to (the purely modal part of) CTL. Let $\mathcal{V}_1 = \{x, y, \ldots\}$ be a set of propositional variables and $\mathcal{V}_2 = \{\mathcal{F}, \ldots\}$ be a set of so-called *recursion* variables. Formulas of RecCTL are given by the following grammar:

$$\varphi ::= q \mid x \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathtt{EX}\varphi \mid \Phi(\varphi, \ldots, \varphi)$$
$$\Phi ::= \mathcal{F} \mid \mathtt{rec}\,\mathcal{F}(x_1, \ldots, x_k).\,\varphi$$

where $x, x_i \in \mathcal{V}_1$, $\mathcal{F} \in \mathcal{V}_2$. A formula derived from φ in this grammar is called *propositional*, one derived from Φ is called *first-order*. Formulas are interpreted over (untimed) LTS $\mathcal{T}$ over some state set $\mathcal{S}$. A propositional formula φ denotes a *predicate* $[\![\varphi]\!]^{\mathcal{T}} \in 2^{\mathcal{S}}$, i.e. a set of states just like any CTL formula does; a first-order formula however denotes a *predicate transformer* $[\![\Phi]\!]^{\mathcal{T}} : 2^{\mathcal{S}} \times \cdots \times 2^{\mathcal{S}} \to 2^{\mathcal{S}}$.

We do not give the details of the formal semantics here. It suffices to note that the recursion operator is interpreted as the least fixpoint in the corresponding complete lattice of first-order, or predicate transformers. For this to work seamlessly, i.e. these fixpoints to exist, we need to guarantee that any variable $\mathcal{F}$ is used only monotonically in φ inside of $\mathtt{rec}\,\mathcal{F}(\vec{x}, \vec{y}).\,\varphi$. The fact that the logic features negation ($\neg$) and application ($\Phi(\varphi_1, \ldots, \varphi_k)$) requires a slightly more involved syntactic criterion for monotonicity. In particular, in order to know whether some variable is used monotonically, it may be required to know this for others as well. For details we refer to the literature [13] or the next section where the machinery is carried out for full Timed Recursive CTL.

An important result on RecCTL to note here, as it will be used later on in Sect. 4, is decidability of its model checking problem.

Proposition 2.2 *([13]). The model checking problem for* RecCTL *over finite LTS is* EXPTIME*-complete.*

3. Timed recursive computation tree logic

We now introduce Timed Recursive CTL, starting by stating its syntax. Given that it is a descendant of RecCTL, we have to deal with negation in order to filter out non-well-formed formulas that cannot be endowed with proper semantics. Once we have given semantics to well-formed formulas, we close with some examples to illustrate what can and cannot be expressed in this logic.

3.1. The formal syntax

3.1.1. Operators of timed recursive CTL

Let *Prop* be a set of atomic propositions and $\mathcal{X}$ be a set of clocks. The syntax of Timed Recursive CTL (TRCTL) is similar to that of RecCTL in that we distinguish between propositional and first-order formulas. We also need two kinds of variables again: first-order variables $\mathcal{V}_2 = \{\mathcal{F}, \mathcal{G}, \ldots\}$ to form recursion anchors and propositional variables $\mathcal{V}_1 = \{x, y, \ldots\}$ for formal parameters of recursive formulas. Formulas are then given by

$$\varphi ::= p \mid \chi \mid x \mid \varphi \wedge \psi \mid \neg\varphi \mid \mathtt{E}(\varphi\ \mathtt{U}_J\ \varphi) \mid \Phi(\varphi, \ldots, \varphi)$$
$$\Phi ::= \mathcal{F} \mid \mathtt{rec}\,\mathcal{F}(x_1, \ldots, x_k).\,\varphi$$

where $p \in \mathit{Prop}$, χ is a clock constraint over $\mathcal{X}$, $k \geq 0$, $x, x_1, \ldots, x_k \in \mathcal{V}_1$, $\mathcal{F} \in \mathcal{V}_2$, and J denotes an interval in $\mathbb{R}^{\geq 0}$ with integer bounds as in the syntax for TCTL. We write $m(\varphi)$ to denote the largest constant that occurs in interval annotations of the Until operators in φ.

Note that CTL features the *Next* operators $Q\,\mathtt{X}$ as well as the *Until* operators $Q\,\mathtt{U}$. The former is missing in TCTL since there is no "next" moment in dense real time. RecCTL, however, seems to feature the *Next* but not the *Until*. This is simply because $Q(\varphi\ \mathtt{U}\ \psi)$ is expressible via $Q\,\mathtt{X}$ using the recursion operator which is stronger than propositional fixpoints, i.e. $Q(\varphi\ \mathtt{U}\ \psi) \equiv (\mathtt{rec}\,\mathcal{F}().\,\psi \vee (\varphi \wedge Q\,\mathtt{X}\mathcal{F}()))()$, written more conveniently as $\mathtt{rec}\,\mathcal{F}.\,\psi \vee (\varphi \wedge Q\,\mathtt{X}\mathcal{F})$, along the lines of the embedding of CTL into the modal μ-calculus. This does not work for the time-bounded *Until* operator any more. Hence, TRCTL features such the *Until* but not the *Next* operator just like TCTL.

Other Boolean and temporal operators are defined in the usual way, for instance $\mathtt{EF}_J\varphi := \mathtt{E}(\mathtt{tt}\ \mathtt{U}_J\ \varphi)$, $\mathtt{AG}_J\varphi := \neg\mathtt{EF}_J\neg\varphi$, etc. and will be used freely henceforth.

This gives rise to fragments of TRCTL which result from the restriction to certain temporal operators or intervals of certain kind only. For example, we write TRCTL[EF] for the set of TRCTL-formulas which use EF (and therefore possibly also AG) as the only temporal operator, and in particular no binary Until operators. Likewise, we write TRCTL[=], TRCTL[$\leq, \geq$]

$$\frac{}{(\emptyset,\emptyset)\vdash p} \qquad \frac{}{(\emptyset,\emptyset)\vdash \chi} \qquad \frac{}{(\{\mathcal{F}\},\emptyset)\vdash \mathcal{F}} \qquad \frac{}{(\{x\},\emptyset)\vdash x}$$

$$\frac{(\mathcal{X},\mathcal{Y})\vdash\varphi \qquad (\mathcal{X}',\mathcal{Y}')\vdash\psi}{(\mathcal{X}\cup\mathcal{X}',\mathcal{Y}\cup\mathcal{Y}')\vdash\varphi\wedge\psi} \qquad \frac{(\mathcal{X},\mathcal{Y})\vdash\varphi}{(\mathcal{Y},\mathcal{X})\vdash\neg\varphi} \qquad \frac{(\mathcal{X},\mathcal{Y})\vdash\varphi \qquad (\mathcal{X}',\mathcal{Y}')\vdash\psi}{(\mathcal{X}\cup\mathcal{X}',\mathcal{Y}\cup\mathcal{Y}')\vdash \mathtt{E}(\varphi\,\mathtt{U}_J\,\psi)}$$

$$\frac{(\mathcal{X},\mathcal{Y})\vdash\varphi \qquad x_1,\dots,x_k\notin\mathcal{Y},\ \ y_1,\dots,y_{k'}\notin\mathcal{X},\ \ \mathcal{F}\notin\mathcal{Y}}{(\mathcal{X}\setminus\{x_1,\dots,x_k,\mathcal{F}\},\mathcal{Y}\setminus\{y_1,\dots,y_{k'}\})\vdash \mathtt{rec}\,\mathcal{F}(x_1,\dots,x_k,y_1,\dots,y_{k'}).\varphi}$$

$$\frac{(\mathcal{X},\mathcal{Y})\vdash\Phi \quad (\mathcal{X}_1,\mathcal{Y}_1)\vdash\varphi_1 \ \cdots\ (\mathcal{X}_k,\mathcal{Y}_k)\vdash\varphi_k \quad (\mathcal{X}'_1,\mathcal{Y}'_1)\vdash\psi_1 \ \cdots\ (\mathcal{X}'_{k'},\mathcal{Y}'_{k'})\vdash\psi_{k'}}{(\mathcal{X}\cup\bigcup_{i=1}^{k}\mathcal{X}_i\cup\bigcup_{j=1}^{k'}\mathcal{Y}'_j,\ \mathcal{Y}\cup\bigcup_{i=1}^{k}\mathcal{Y}_i\cup\bigcup_{j=1}^{k'}\mathcal{X}'_j)\vdash\Phi(\varphi_1,\dots,\varphi_k,\psi_1,\dots,\psi_{k'}).\varphi}$$

Fig. 1. The rules for establishing well-formedness of a TRCTL formula.

and TRCTL[$<,>$] for the fragment in which every interval is formed using the corresponding operators only. We will use index notation like TRCTL$_k$ to denote the fragment of TRCTL whose formulas are constructed over a set of clocks $\mathcal{X}$ of fixed size k. We combine these notations into something like TRCTL$_1$[EF, $=$] for instance, restricting both the use of temporal operators and intervals at the same time, and allowing a single clock to occur in the clock constraints in formulas as well as the timed automata that they get interpreted over. So, for example, $\mathtt{EF}_{=4}(\mathtt{x}=2)$ is a formula of this fragment, while $\mathtt{EF}_{\leq 4}(\mathtt{x}=2)$ is not.

3.1.2. Well-formed formulas

Not every formula generated by the formal syntax as introduced above is well-formed. For instance, when a recursion formula has k formal parameters as in the formula $\Phi=\mathtt{rec}\,\mathcal{F}(x_1,\dots,x_k).\varphi$, it should only be applied to a tuple of k arguments as in $\Phi(\varphi_1,\dots,\varphi_k)$. The same goes for any subformula of the form $\mathcal{F}(\psi_1,\dots,\psi_k)$. Violations of this are, of course, quite easily spotted, so we will not formalise this requirement here.

A more difficult issue is that of monotonicity: The recursion operator in TRCTL is explained via fixpoints in complete lattices, and this requires the defining formula of a recursion to be monotonic in its arguments. Since it is possible to define non-monotonic functions in TRCTL, e.g. via $\mathtt{rec}\,\mathcal{F}(x).\neg x$, tracking whether a complex formula is monotonic in a given argument is difficult. The typing system in Fig. 1 does this. A statement of the form $(\mathcal{X},\mathcal{Y})\vdash\varphi$ says that the variables that occur positively in φ are exactly those in $\mathcal{X}$, and the variables that occur negatively in φ are exactly those in $\mathcal{Y}$. Positive occurrence is a generalisation of the well-known criterion to occur only under an even number of negations, and, hence, entails that the formula in question is monotonic in the variable in question.

The rules then formalise the behaviour of the intended semantics of TRCTL formulas w.r.t. monotonicity. For example, the penultimate rule formalises that, in a formula of the form $\mathtt{rec}\,\mathcal{F}(x_1,\dots,x_k,y_1,\dots,y_{k'}).\varphi$, the variables $x_1,\dots,x_k,y_1,\dots,x_{k'}$ as well as the recursion variable $\mathcal{F}$ do not occur freely. However, in φ, the recursion variable $\mathcal{F}$ may not occur negatively, and neither can the variables $x_1,\dots,x_k$ identified as parameters in which φ is monotonic. Conversely, $y_1,\dots,y_{k'}$ may not occur positively, since φ is antitonic in these.

Note that this version of the rules assumes that all variables that occur monotonically in a recursive definition appear before all variables that occur antitonically. This can, of course, be relaxed at the cost of more notation; we refrain from doing that here to keep this already somewhat unwieldy definition readable. For a detailed discussion of the typing system, including an in-depth explanation of its rules, see [13] where the notion of well-formedness is made precise for RecCTL, an untimed version of TRCTL.

Definition 3.1. We call a formula φ of TRCTL without free variables *well-formed* if the statement $(\emptyset,\emptyset)\vdash\varphi$ is derivable using the rules shown in Fig. 1.

3.1.3. Vectorial form

The semantics of the recursion operator will be explained using least fixpoints in complete function lattices. This makes the Bekič Lemma [24] available which allows formulas with mutual dependencies between recursion variables to be written down in a more readable form. A formula in *vectorial form*, (see e.g. [25] for its use in the modal μ-calculus) is a

$$\mathtt{rec}_i\begin{pmatrix}\mathcal{F}_1(x_1,\dots,x_k) & . & \varphi_1\\ & \vdots & \\ \mathcal{F}_n(x_1,\dots,x_k) & . & \varphi_n\end{pmatrix}(\psi_1,\dots,\psi_k).$$

Informally, this defines not just one but several functions $\mathcal{F}_1,\dots,\mathcal{F}_n$ which may all depend on each other in a mutually recursive way formalised in the φ_j's. In the end, the function named by $\mathcal{F}_i$ is applied to the initial arguments $\psi_1,\dots,\psi_k$.

3.2. The formal semantics

As with TCTL, (propositional) well-formed formulas of TRCTL are interpreted in states of a TLTS $\mathcal{T} = (\mathcal{S}, \to, s_0, \lambda)$. In fact, it suffices to extend the semantics of TCTL to those operators (propositional variables and first-order formulas) which do not already occur in the syntax of TCTL. Due to the presence of variables, we need variable interpretations ϑ in order to explain the meaning of a formula inductively. Such a ϑ maps propositional variables to sets of states, $\vartheta(x) \in 2^{\mathcal{S}}$ for $x \in \mathcal{V}_1$, and first-order variables to functions of corresponding arity over these: $\vartheta(\mathcal{F}) : 2^{\mathcal{S}} \times \ldots \times 2^{\mathcal{S}} \to 2^{\mathcal{S}}$. These functions form a complete Boolean lattice ordered pointwise, hence least fixpoints of monotone functionals mapping one such function to another exist due to the Knaster-Tarski Theorem [26]. These fixpoints are used to explain the meaning of the recursion operator. For details, we refer to the exposition on RecCTL [13] or on HFL [7] that this idea goes back to – the only difference is that there, $\mathcal{S}$ is the state space of an untimed LTS rather than a TLTS.

A propositional formula φ gives rise to a set $[\![\varphi]\!]^{\mathcal{T}}_{\vartheta}$ of states that satisfy it under the variable interpretation ϑ, and similarly for first-order formulas and corresponding first-order functions. The semantics is defined as follows. The clauses presented for $\varphi \in \mathrm{TCTL}$ apply here as well under the provision that each $[\![\cdot]\!]^{\mathcal{T}}$ is replaced by $[\![\cdot]\!]^{\mathcal{T}}_{\vartheta}$. Additionally,

$$[\![x]\!]^{\mathcal{T}}_{\vartheta}\,\vartheta(x) \quad \text{for } x \in \mathcal{V}_1 \quad \text{and} \quad [\![\Phi(\varphi_1, \ldots, \varphi_k)]\!]^{\mathcal{T}}_{\vartheta}\,[\![\Phi]\!]^{\mathcal{T}}_{\vartheta}([\![\varphi_1]\!]^{\mathcal{T}}_{\vartheta}, \ldots, [\![\varphi_k]\!]^{\mathcal{T}}_{\vartheta})$$

for propositional formulas, while for first-order formulas we set $[\![\mathcal{F}]\!]^{\mathcal{T}}_{\vartheta} := \vartheta(\mathcal{F})$ if $\mathcal{F} \in \mathcal{V}_2$ and

$$[\![\mathtt{rec}\,\mathcal{F}(x_1, \ldots, x_k).\,\varphi]\!]^{\mathcal{T}}_{\vartheta} := \bigsqcap \{f : (2^{\mathcal{S}})^k \to 2^{\mathcal{S}} \mid \forall S_1, \ldots, S_k : [\![\varphi]\!]^{\mathcal{T}}_{\vartheta[\mathcal{F} \mapsto f, x_1 \mapsto S_1, \ldots, x_k \mapsto S_k]} \subseteq f(S_1, \ldots, S_k)\}$$

where $\sqcap$ denotes the point-wise intersection for functions: $(f \sqcap g)(S) := f(S) \cap g(S)$.

3.3. Examples

We illustrate the use of the recursion operator in TRCTL to form structurally complex properties which cannot be expressed in TCTL. We refer to [13] for more exposition regarding RecCTL. It is helpful, though, to imagine the recursive formulas to be unrolled so that new arguments are being built and these to be plugged in for the formal parameters.

Example 3.2. Consider $\varphi_{\mathsf{ag}} := \big(\mathtt{rec}\,\mathcal{F}(x, y).\,(x \wedge \neg y) \vee \mathcal{F}(\mathtt{AF}_{\leq 3}x, \mathtt{AF}_{\leq 2}y)\big)(p, p)$. Unrolling of the recursion shows that it is equivalent to

$$\bigvee_{i \geq 0} \underbrace{\mathtt{AF}_{\leq 3}\mathtt{AF}_{\leq 3} \ldots \mathtt{AF}_{\leq 3}}_{i \text{ times}}\, p \wedge \neg \underbrace{\mathtt{AF}_{\leq 2}\mathtt{AF}_{\leq 2} \ldots \mathtt{AF}_{\leq 2}}_{i \text{ times}}\, p$$

stating "there is an i such that on all paths we see i occurrences of p in distances of at most 3 seconds, but not in distances of at most 2 seconds." Negating this to $\neg\varphi_{\mathsf{ag}}$ then formalises "whenever it is possible to see p in distances of 3 seconds i times on a path, then it is also possible to do so in distances of 2 seconds on some path." This is inspired by the formalisation of assume-guarantee properties in HFL [7].

Example 3.3. Note that the context-free grammar G with productions

$$F_1 \to F_2F_3 \;, \quad F_2 \to \mathsf{out} \mid \mathsf{in}F_2F_2 \;, \quad F_3 \to \varepsilon \mid \mathsf{in}F_3 \mid \mathsf{out}F_3$$

generates the set of all $\{\mathsf{in}, \mathsf{out}\}$-sequences such that some prefix contains more out's than in's. It can be seen as the set of all finite computations in which a buffer underflow occurs. Now consider the TRCTL formula

$$\varphi_{\mathsf{buf}} := \mathtt{rec}_1 \begin{pmatrix} \mathcal{F}_1(x) & . & \mathcal{F}_2(\mathcal{F}_3(x)) \\ \mathcal{F}_2(x) & . & \mathtt{E}(p_{\mathsf{out}}\,\mathtt{U}_{\geq 1}\,x) \vee \mathtt{E}(p_{\mathsf{in}}\,\mathtt{U}_{\geq 1}\,\mathcal{F}_2(\mathcal{F}_2(x))) \\ \mathcal{F}_3(x) & . & x \vee \mathtt{E}(p_{\mathsf{in}}\,\mathtt{U}_{\geq 1}\,\mathcal{F}_3(x)) \vee \mathtt{E}(p_{\mathsf{out}}\,\mathtt{U}_{\geq 1}\,\mathcal{F}_3(x)) \end{pmatrix} (\mathtt{tt})\,.$$

It states that there is a path forming a buffer underflow, provided that consecutive traversal of states satisfying p_{in} or p_{out} for at least 1 second are taken as input/output actions for the buffer. Then $\neg\varphi_{\mathsf{buf}}$ formalises absence of such underflows under this interpretation.

4. Upper bounds for model checking

We show that the model checking problem for TRCTL is 2-EXPTIME-complete. We begin with the upper bound based on a standard untiming construction called the region graph construction [17], which turns a TLTS arising from a TA $\mathcal{A}$ into a finite untimed LTS known as the *region graph* $\mathcal{R}_{\mathcal{A}}$. This construction and its derivatives are often used in decidability proofs for decision problems on TA. Let $\varphi \in \mathrm{TRCTL}$ and $\mathcal{A}$ be a TA, both over the same sets of clocks $\mathcal{X}$ and atomic propositions *Prop*. In the following we only consider TLTS $\mathcal{T}_{\mathcal{A}}$ that arise from some TA $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda_{\mathcal{A}})$.

4.1. The region abstraction

The region abstraction is a mapping of such TLTS into finite LTS. It is based on an equivalence relation $\simeq_m$, for $m \in \mathbb{N}$, on clock evaluations defined as (see also [22], Def. 9.42): $\eta \simeq_m \eta'$ iff

$$\begin{aligned}&\text{for all } x \in \mathcal{X} : \eta(x) > m \text{ and } \eta'(x) > m\\ &\quad \text{or } \lfloor \eta(x) \rfloor = \lfloor \eta'(x) \rfloor \text{ and } \mathit{frac}(\eta(x)) = 0 \Leftrightarrow \mathit{frac}(\eta'(x)) = 0\\ &\quad\quad \text{and for all } y \in \mathcal{X} \text{ with } \eta(y) \leq m \text{ and } \eta'(y) \leq m :\\ &\quad\quad\quad \mathit{frac}(\eta(x)) \leq \mathit{frac}(\eta(y)) \Leftrightarrow \mathit{frac}(\eta'(x)) \leq \mathit{frac}(\eta'(y)).\end{aligned}$$

Here, $\mathit{frac}(r)$ denotes the fractional part of a real number. The above definition makes clock evaluations equivalent iff, for each clock, (I) either both clocks have a value bigger than m, or (II) they compare in the same way w.r.t. all integers less than m and, moreover, the passage of time will have equivalent evaluations reach the next integral value first for the same clock. It is not hard to verify that $\simeq_m$ is indeed an equivalence relation for any m. An equivalence class in this equivalence relation is also called a *region*.

The equivalence relation is lifted to states of the TLTS $\mathcal{T}_{\mathcal{A}}$ in the most straightforward way by setting

$$(\ell, \eta) \simeq_m (\ell', \eta') \quad \text{iff} \quad \ell = \ell' \text{ and } \eta \simeq_m \eta' .$$

We write $[\eta]_m$ for the equivalence class of η under $\simeq_m$ and likewise for $[(\ell, \eta)]_m$ which we usually write as $(\ell, [\eta]_m)$ since they are indeed the same. When m is clear from the context we may also drop it and simply write $[\eta]$ or $(\ell, [\eta])$, respectively.

This is not only an equivalence relation on the state space of $\mathcal{T}_{\mathcal{A}}$ but in fact even a congruence w.r.t. the labelling and discrete and delay transitions when $m \geq m(\mathcal{A})$. This is what makes it usable in order to abstract the uncountable state space of $\mathcal{T}_{\mathcal{A}}$ into a finite discrete state space. However, note that a delay transition in a TLTS may cross several regions at once, for example if delaying by one time unit or more. This has to be made explicit in the transition structure of the region graph. Hence, for each region $[\eta]_m$, we define a unique *successor region* via

- $\mathit{suc}([\eta]_m) = [\eta]_m$ if $\eta(\mathrm{x}) > m$ for all $\mathrm{x} \in \mathcal{X}$,
- $\mathit{suc}([\eta]_m) = [\eta']_m$ iff there is $d \in \mathbb{R}^{\geq 0}$ such that $\eta + d = \eta'$, and $\eta + d' \in [\eta]_m \cup [\eta']_m$ for all $0 < d' < d$, and $[\eta]_m \neq [\eta']_m$.

This simply formalises the notion that $\mathit{suc}([\eta]_m)$ is either $[\eta]_m$ itself in case that all clocks have values bigger than m, or that $\mathit{suc}([\eta]_m)$ is the first region different from $[\eta]_m$ that one enters if time passes.

Let Ξ be a finite set of clock constraints over the clocks $\mathcal{X}$ that the timed automaton $\mathcal{A}$ is defined over. These clock constraints will be made visible as additional propositions in the construction. The *region graph* of the TA $\mathcal{A}$ with additional clock constraints in Ξ, written $\mathcal{R}^{\Xi}_{\mathcal{A}}$, is the LTS (over propositions $\mathit{Prop} \cup \Xi$) obtained as the quotient of $\mathcal{T}_{\mathcal{A}}$ under the congruence relation $\simeq_m$ (with $m = m(\mathcal{A})$), together with an additional collapse of delay transitions for different delays into a single "*some-delay*" value τ that connects a region with its successor region. The components of the region graph are as follows.

- The state space is $\{(\ell, [\eta]_m) \in L \times (\mathcal{X} \to \mathbb{R}^{\geq 0})/_{\simeq_m} \mid \eta \models \iota(\ell)\}$. The initial state is $(\ell_0, [\eta_0]_m)$.
- Discrete transitions from one state to another state are carried through, i.e. we have $(\ell, [\eta]_m) \to (\ell', [\eta']_m)$ iff $\eta \simeq_m \eta'$ and $(\ell, \eta) \to (\ell', \eta')$.
- Delay transitions always lead to the successor region, i.e. $(\ell, [\eta]_m) \to (\ell', [\eta']_m)$ iff $\ell = \ell'$ and $[\eta']_m = \mathit{suc}([\eta]_m)$.
- The propositional labelling not only assigns atomic propositions to states via $p \in \lambda(\ell, [\eta])$ iff $p \in \lambda_{\mathcal{A}}(\ell)$ for any $p \in \mathit{Prop}$, but also interprets any clock constraint $\chi \in \Xi$ as an atomic proposition in the region graph via $\chi \in \lambda(\ell, [\eta])$ iff $(\ell, \eta) \models \chi$.

Proposition 4.1 *([17]). Let $\mathcal{A}$ be a TA over n clocks with ℓ locations and of index m, and Ξ be a set of clock constraints over these clocks. Then $\mathcal{R}^{\Xi}_{\mathcal{A}}$ is an (untimed) LTS of size $\ell \cdot 2^{\mathcal{O}(n(\log n + \log m))} \cdot |\Xi|$, i.e. exponential in $|\mathcal{A}|$, and there is a trace $s_0 \xrightarrow{d_0} s_1 \xrightarrow{d_1} \dots$ in $\mathcal{T}_{\mathcal{A}}$ iff there is a path $[s_0] \to [s_1] \to \dots$ in $\mathcal{R}^{\Xi}_{\mathcal{A}}$.*

4.2. Elimination of interval bounds

We assume that there is some clock $\mathrm{z} \notin \mathcal{X}$ which is mentioned neither in $\mathcal{A}$ nor in φ. This clock z will be used to remove intervals from the temporal operators in φ, making the passing of time explicit. The intuitive trick is to replace, e.g. a subformula $\mathtt{EF}_{[c,d]}\psi$ by $\mathtt{EF}(\mathrm{z} \geq c \wedge \mathrm{z} \leq d \wedge \psi)$ thus making the moment explicit at which a time point in the interval $[c, d]$ is reached. This is of course not sound in general, as nothing guarantees that z has the value 0 at the beginning of the evaluation of this formula. This cannot be enforced in the formula; note that $\mathrm{z} = 0 \wedge \mathtt{EF}(\mathrm{z} \geq c \wedge \mathrm{z} \leq d \wedge \psi)$ does not

express the desired property for instance. In fact, the required combinator between $\mathtt{z} = 0$ and the rest of the formula is an *and-then* in the sense that $\mathtt{z}$ is to be reset and then the formula is supposed to hold, which is not Boolean in nature. This can be realised by constructions in the model in a slightly non-standard way (see [22], Thm. 9.37). For this we use an additional proposition r that is not mentioned anywhere in φ or $\mathcal{A}$.

First we consider the corresponding amendment of the formula φ. Let $\varphi^{\mathtt{z}}$ result from it by replacing every subformula

- $\mathtt{E}(\psi_1 \,\mathtt{U}_J\, \psi_2)$ by $\mathtt{EX}(r \wedge \mathtt{EXE}((\neg r \wedge \psi_1^{\mathtt{z}})\,\mathtt{U}\,(\neg r \wedge \mathtt{z} \in J \wedge \psi_2^{\mathtt{z}})))$, and
- $\mathtt{A}(\psi_1 \,\mathtt{U}_J\, \psi_2)$ by $\mathtt{EX}(r \wedge \mathtt{EXA}((\neg r \to \psi_1^{\mathtt{z}})\,\mathtt{U}\,(\neg r \to \mathtt{z} \in J \wedge \psi_2^{\mathtt{z}})))$.

Note that $\varphi^{\mathtt{z}}$ contains additional clock constraints in comparison to φ, resulting from the elimination of intervals in the temporal operators.

Let $\mathcal{R} := \mathcal{R}_{\mathcal{A}}^{\Xi}$ where Ξ is the set of all clock constraints occurring in $\varphi^{\mathtt{z}}$. Let $\mathcal{R}'$ result from it by adding, for each state $(\ell, [\eta])$ a new state $(\ell, [\eta])'$ which is labelled with the proposition r only, and has transitions

$$(\ell, [\eta]) \to (\ell, [\eta])' \to (\ell, [\eta|_{\{\mathtt{z}\}}])\ .$$

This has introduced new traces in this region graph: at any moment, it is now possible to reset clock $\mathtt{z}$, and then continue some original trace. Moreover, the resetting of $\mathtt{z}$ becomes visible through the traversal of a state that satisfies r. Since $\mathtt{z}$ is not used in $\mathcal{A}$, this is the only way that it is being reset. Finally, during such a reset of clock $\mathtt{z}$, no time elapses.

The following forms the basis of an exponential reduction of TRCTL model checking to RecCTL model checking.

Lemma 4.2. *Let $\mathcal{A}$ be a TA, $\varphi \in$ TRCTL, and $\mathcal{R}'$, $\varphi^{\mathtt{z}}$ be as defined above.*

a) *$\varphi^{\mathtt{z}}$ is a formula of (untimed)* RecCTL *and is constructible in time $\mathcal{O}(|\varphi|)$.*
b) *$\mathcal{R}'$ is an (untimed) LTS of size at most (singly) exponential in $|\mathcal{A}|$ and linear in $|\varphi|$ and also constructible in such time.*
c) *$\mathcal{T}_{\mathcal{A}} \models \varphi$ iff $\mathcal{R}' \models \varphi^{\mathtt{z}}$.*

Proof. Part (a) is easily verified. Part (b) follows directly from Proposition 4.1. It remains to show part (c), which is done by an induction on the structure of φ.

In order to deal with variable interpretations, define, for a variable interpretation ϑ on $\mathcal{T}_{\mathcal{A}}$, a corresponding variable interpretation ϑ on $\mathcal{R}'$ via $\vartheta(x) = \{(l, [\eta]) \mid (l, \eta) \in \vartheta(X)\}$. We can then show by induction that

$$(\ell, \eta) \in [\![\psi]\!]_{\vartheta}^{\mathcal{T}_{\mathcal{A}}} \text{ iff } (\ell, [\eta]) \in [\![\psi^{\mathtt{z}}]\!]_{\vartheta}^{\mathcal{R}'}$$

for all subformulas ψ of φ. Note that this implies that, similarly to TCTL, the logic TRCTL cannot distinguish states that are equivalent under the region abstraction.

The interesting cases are that of an until formula, i.e. one of the form $\mathtt{E}(\psi_1 \,\mathtt{U}_J\, \psi_2)$, and that of recursion and application. The first case follows via a standard construction to integrate path constraints into the region graph. First, observe that r holds exactly at the extra states of the form $(\ell, [\eta])'$, i.e. those after which $\mathtt{z}$ is reset. Hence,

$$(\ell, [\eta]) \in [\![\mathtt{EX}(r \wedge \mathtt{EXE}((\neg r \wedge \psi_1^{\mathtt{z}})\,\mathtt{U}\,(\neg r \wedge \mathtt{z} \in J \wedge \psi_2^{\mathtt{z}})))]\!]_{\vartheta}^{\mathcal{R}'}$$

iff

$$(\ell, [\eta|_{\{\mathtt{z}\}}]) \in [\![\mathtt{E}((\neg r \wedge \psi_1^{\mathtt{z}})\,\mathtt{U}\,(\neg r \wedge \mathtt{z} \in J \wedge \psi_2^{\mathtt{z}}))]\!]_{\vartheta}^{\mathcal{R}'}$$

for all (timed) variable interpretations ϑ, and the analogue holds for universally quantified $\mathtt{U}$-formulas. By Proposition 4.1 we know that there is a trace $s_0 \xrightarrow{d_0} s_1 \xrightarrow{d_1} \dots$ in $\mathcal{T}_{\mathcal{A}}$ iff there is a path $[s_0] \to [s_1] \to \dots$ in $\mathcal{R}_{\mathcal{A}}^{\Xi}$, and this holds iff there is such a path in $\mathcal{R}'$ without going through states labelled r. It follows that

$$(\ell, \eta) \in [\![\mathtt{E}(\psi_1 \,\mathtt{U}_J\, \psi_2)]\!]_{\vartheta}^{\mathcal{T}_{\mathcal{A}}} \quad \text{iff} \quad (\ell, [\eta]) \in [\![\mathtt{EX}(r \wedge \mathtt{EXE}((\neg r \wedge \psi_1^{\mathtt{z}})\,\mathtt{U}\,(\neg r \wedge \mathtt{z} \in J \wedge \psi_2)))]\!]_{\vartheta}^{\mathcal{R}'}.$$

The second case is that of recursion, i.e. that of a subformula of the form $\Phi(\psi_1, \dots, \psi_k)$ with $\Phi = \mathtt{rec}\, \mathcal{F}(x_1, \dots, x_k).\, \psi'$. Recall that the semantics of ψ is defined as

$$\bigsqcap \{f : (2^{\mathcal{S}})^k \to 2^{\mathcal{S}} \mid \forall S_1, \dots, S_k : [\![\psi']\!]_{\vartheta[\mathcal{F} \mapsto f, x_1 \mapsto S_1, \dots, x_k \mapsto S_k]}^{\mathcal{T}_{\mathcal{A}}} \subseteq f(S_1, \dots, S_k)\}$$

using the Knaster-Tarski Theorem. Here we make use of the Kleene Fixpoint Theorem which states that this least fixpoint can equivalently be obtained as the limit of a sequence of functions that is defined via

$$f_0 = (S_1, \dots, S_k) \mapsto \emptyset$$
$$f_{i+1} = (S_1, \dots, S_k) \mapsto [\![\psi']\!]_{\vartheta[\mathcal{F} \mapsto f_i, x_1 \mapsto S_1, \dots, x_k \mapsto S_k]}^{\mathcal{T}_{\mathcal{A}}}.$$

This characterisation also holds for RecCTL by replacing $\mathcal{T}_\mathcal{A}$ by $\mathcal{R}'$ and ψ' by $\psi'^{\mathtt{z}}$ which yields a sequence of functions defined as

$$f'_0 = (S_1, \dots, S_k) \mapsto \emptyset$$
$$f'_{i+1} = (S_1, \dots, S_k) \mapsto [\![\psi'^{\mathtt{z}}]\!]^{\mathcal{R}'}_{\vartheta[\mathcal{F} \mapsto f_i, x_1 \mapsto S_1, \dots, x_k \mapsto S_k]}$$

which can easily be seen to stabilise after finitely many steps since $\mathcal{R}'$ is finite. Using the induction hypothesis, we obtain that

$$(\ell, \eta) \in f_i([\![\psi_1]\!]^{\mathcal{T}_\mathcal{A}}_\vartheta, \dots, [\![\psi_k]\!]^{\mathcal{T}_\mathcal{A}}_\vartheta) \text{ iff } (\ell, [\eta]) \in f'_i([\![\psi_1^{\mathtt{z}}]\!]^{\mathcal{R}'}_\vartheta, \dots, [\![\psi_k^{\mathtt{z}}]\!]^{\mathcal{R}'}_\vartheta)$$

for all $i \in \mathbb{N}$ and for all variable interpretations ϑ. Since, for some i, we have that $f'_i = f'_{i+1}$, the desired result follows. □

It follows that we obtain that fixpoint iteration in the sense of the Kleene's Fixpoint Theorem also stabilises for TRCTL after finitely many steps, even though the LTS generated by a TA is not finite.

4.3. The reduction

Theorem 4.3. *The model checking problem for* TRCTL *over TA is decidable in* 2-EXPTIME.

Proof. Let a TA $\mathcal{A}$ and a TRCTL formula φ be given. To check whether $\mathcal{T}_\mathcal{A} \models \varphi$ holds, first construct $\mathcal{R}'$ and $\varphi^{\mathtt{z}}$ as described above. According to Lemma 4.2, this can be done in exponential time, and it suffices to check whether or not $\mathcal{R}' \models \varphi^{\mathtt{z}}$ holds. According to Proposition 2.2, the latter can be solved in exponential time. Altogether, this gives a doubly exponential upper bound on the time complexity of model checking TRCTL over TA. □

5. Lower bounds for model checking

We now proceed with the lower bound proof. Towards this, we characterise doubly-exponential time in Proposition 5.1. We then present a generic set of minimal operations on (representations of) large numbers that need to be available to extract a hardness proof based on Proposition 5.1. We then show that this set of minimal operations can be realised in a number of fragments of TRCTL, i.e. restrictions of TRCTL to a minimal set of operators such as TRCTL[$\mathtt{EF}, =$]. Moreover, we can carry out these hardness proofs over a fixed TA with one clock only, which yields hardness already for the expression complexity of TRCTL and which is optimal in the usage of clocks (note that the 0-clock setting is RecCTL, whose model checking problem is known to be in EXPTIME).

5.1. Doubly exponential time complexity

The lower bound for the complexity model checking problem of TRCTL is established, as usual, by a polynomial reduction from a problem that is already known to be 2-EXPTIME-hard.

5.1.1. Witnesses for doubly exponential time complexity

The generic candidate of a 2-EXPTIME-hard problem is the word problem for deterministic doubly exponentially time-bounded Turing machines. Such reductions to decision problems on temporal logics typically encode witnesses like Turing machine runs as temporal structures like traces for linear-time logics, or trees for branching-time logics. This creates a slight mismatch here: the standard witness for acceptance of a word by a deterministic Turing machine is an inherently linear structure. Properties expressed in CTL-like branching-time logics generally look for branching structures in models, though. However, luckily there is a characterisation of deterministic time-bounded complexity classes via alternating Turing machines. In particular, the class 2-EXPTIME coincides with the class of problems solved by an alternating exponentially space-bounded Turing machine [20].

This shows that besides the traditional and straightforward characterisation of problems solvable in 2-EXPTIME through the existence of linear runs (of particular nature because of the underlying determinism), there is also a characterisation via the existence of tree-like structures in principle. While we do not make use of the word problem for alternating exponential time-bounded Turing machines explicitly, we can use such an alternating, tree-like characterisation using exponential memory. This characterisation is the square table of doubly-exponential size introduced in [20] that witnesses an accepting run of a 2-EXPTIME machine which, when explored locally, can fit into alternating exponential space. This reformulation allows us to focus on the source of this high complexity coming from the power of the operators in the logic, separating this from the issues on real-time in the underlying model, which are used to generated storage of exponential size.

It is standard to reduce the word problem for Turing machines to the empty-word problem, asking whether a given Turing machine accepts the empty word (within given resource bounds), as it is easy to construct, from a machine $\mathcal{M}$ and a word w, a machine $\mathcal{M}_w$ which first replaces its input by w and then proceeds with the computation of $\mathcal{M}$ on it. Moreover, it is equally standard to assume that acceptance at the end of a computation is only signalled after the head has

been moved to the leftmost position on it, to remain there until a possible time bound has run out. This standardisation of a final configuration makes a reduction technically easier.

Finally, since we consider computational effort modulo polynomials only, we can assume that the time bound for a deterministic machine under consideration is not only some $2^{2^{p(n)}}$ for a polynomial $p(n)$ but just 2^{2^n}. In fact, for technical reasons we assume it to be $2^{2^n} - 2$, which is of course still possible within the limits of polynomial reductions.

5.1.2. Deterministic Turing machines

A Deterministic Turing Machine (DTM) is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta, q_{\text{acc}})$, where Q is a finite set of states containing the initial and accepting states q_0 and q_{acc}, Σ is the input alphabet, $\Gamma \supseteq \Sigma$ is the tape alphabet containing a special symbol $\square \in \Gamma \setminus \Sigma$. We assume the existence of a special boundary symbol # that is not contained in Γ. Finally, $\delta \subseteq Q \times \Gamma \to Q \times \Gamma \times \{\mathsf{L}, \mathsf{R}, \mathsf{N}\}$ is the transition function.

Let $\hat{\Gamma} := \Gamma \cup (Q \times \Gamma) \cup \{\#\}$. Let $f : \mathbb{N} \to \mathbb{N}$. The unique $f(n)$-time-bounded computation of $\mathcal{M}$ on the empty input can be represented by a square, containing $f(n)$ rows, representing time, each of which contains $f(n)$ symbols from $\hat{\Gamma}$, representing a configuration, or space. Each row is of the form $\#w\#$ for some $w \in (\hat{\Gamma} \setminus \{\#\})^{f(n)-2}$ containing exactly one symbol from $Q \times \Gamma$; the bottom row is $\#(q_0, \square)\square^{f(n)-3}\#$, and the top row is of the form $\#(q_{\text{acc}}, \square)w\#$ for some $w \in \hat{\Gamma}^{f(n)-3}$. Thus, the square of dimension $f(n) \times f(n)$ represents, from bottom to top, the successive configurations of the DTM, padded by the boundary symbols on both sides.

$\mathcal{M}$'s transition function δ gives rise to a relation $\hat{\delta} \subseteq \hat{\Gamma}$ such that $(y_1, y_2, y_3, x) \in \hat{\delta}$ iff whenever y_1, y_2, y_3 are consecutive symbols in row t at positions $s-1, s, s+1$, then x is the symbol at position s in row $t+1$.

5.1.3. Certificates for doubly exponential time complexity

An $f(n)$-*certificate* (for $\mathcal{M}$ and given n) is a set of mutually recursive predicates $Cert_a : [f(n)] \times [f(n)] \to \{\top, \bot\}$, one for each $a \in \hat{\Gamma}$ with the following properties. Intuitively, $Cert_a(t, s)$ holds true iff the s-th symbol in the t-th configuration of the unique computation of $\mathcal{M}$ on the empty input is a. Formally, this certificate must satisfy the following properties.

- $Cert_{(q_{\text{acc}}, \square)}(f(n) - 1, 0)$ holds true.
- For all $t \in \{1, \dots, f(n) - 1\}$, $s \in \{1, \dots, f(n) - 2\}$ and $a \in \hat{\Gamma} \setminus \{\#\}$ with $Cert_a(t, s)$ there are $b_1, b_2, b_3 \in \hat{\Gamma}$ with $(b_1, b_2, b_3, a) \in \hat{\delta}$ such that

$$Cert_{b_1}(t-1, s-1) \wedge Cert_{b_2}(t-1, s) \wedge Cert_{b_3}(t-1, s+1)$$

  holds true.
- For all $t \in \{0, \dots, f(n) - 1\}$, $s \in \{0, f(n) - 1\}$, we have $Cert_a(t, s)$ iff $a = \#$.
- $Cert_a(0, 1)$ iff $a = (q_0, \square)$, and for all $s = 2, \dots, f(n) - 2$: $Cert_a(0, s)$ iff $a = \square$.

Note that the last two clauses fix the values of a in $Cert_a(t, s)$ uniquely for the left, lower und right edge of the square defined by the coordinates t, s, and determinism of the TM $\mathcal{A}$ then fixes the values at the inner coordinates uniquely as well.

As mentioned before, the above characterisation of acceptance in deterministic time-bounded Turing machines is taken from the construction of their simulation by alternating space-bounded Turing machines in [20]. It can therefore be used to establish a generic 2-EXPTIME-hardness result.

Proposition 5.1. *It is* 2-EXPTIME*-hard to decide, given a DTM $\mathcal{M}$ and an $n \in \mathbb{N}$ encoded unarily, whether or not there is a 2^{2^n}-certificate for $\mathcal{M}$ and n in the sense above.*

Proof. This can be seen by a polynomial-time reduction from the word problem for any DTM $\mathcal{M}'$ deciding a 2-EXPTIME-complete problem. Given $\mathcal{M}'$ and an input w, clearly $\mathcal{M}'_w$, which first replaces the input by w and then behaves like $\mathcal{M}$, can be computed in polynomial time. This replaces the word problem for $\mathcal{M}'$ by the empty-word problem for $\mathcal{M}'_w$. $\mathcal{M}'_w$ accepts the empty word, and, hence, $\mathcal{M}$ accepts w, iff there is a 2^{2^n}-certificate for $\mathcal{M}'_w$: Clearly, an accepting run of $\mathcal{M}$ on the empty word gives rise to a 2^{2^n}-certificate by simply writing down the configurations of the run, delimited by #, below each other. On the other hand, a 2^{2^n}-certificate for $\mathcal{M}'$ is a proof that an accepting run of $\mathcal{M}'$ on the empty word exists, as the values of $Cert_a(t, s)$ for fixed t spell out the t-th configuration of the run. It is easily verified that if the values of the $Cert_a(t, s)$ give rise to some configuration, then the values of the $Cert_a(t+1, s)$ spell out the unique successor configuration. □

The prerequisite of n being given in unary encoding rather than the perhaps more expected binary encoding is not a trick to disguise an exponential blow-up as a polynomial one. Note that, in the reduction sketched above from the word problem with input consisting of a DTM $\mathcal{M}$ and a word w to the empty-word problem for a simulating DTM $\mathcal{M}_w$, the time needed for $\mathcal{M}_w$'s computation on the empty word is largely determined by the time needed for $\mathcal{M}$'s computation on w on length $|w|$. The parameter n given in the formulation of the empty-word problem above can be seen as the remains

of the input word w which has been factored into the machine in order to make the reduction technically simpler. But such a parameter is still needed in order to facilitate the time bound in a sensible way. So n can be seen as the remains of w in terms of its length, and a unary encoding of n guarantees that an $f(n)$-time bound in the empty-word problem corresponds to an $f(n)$-time bound in the original word problem. If n was encoded in binary, we would have to consider triply exponential time bounds in the empty-word problem and would additionally have to separate the inputs $\mathcal{M}$ and w in the word problem. So unary encoding is the natural choice here.

5.2. A generic template for a doubly exponential lower time-bound

A doubly exponential lower bound for model checking TRCTL can be obtained by a polynomial reduction from the problem stated in Proposition 5.1, namely deciding the existence of a 2^{2^n}-certificate (for the empty-word problem) for a given DTM. To establish this we would need to construct, given such a DTM $\mathcal{M}$ and an $n \in \mathbb{N}$, a TA $\mathcal{A}_{\mathcal{M},n}$ and a TRCTL formula $\varphi_{\mathcal{M},n}$ each of polynomial size in $|\mathcal{M}|$ and n, such that $\mathcal{A}_{\mathcal{M},n} \models \varphi_{\mathcal{M},n}$ iff there is a 2^{2^n}-certificate for $\mathcal{M}$ and n. However, we aim to maximise effects in the sense of establishing lower bounds for some small fragments. In order to do so, we will provide a generic template for 2-EXPTIME-hardness which, as the following will show, is mainly due to the higher-order nature of TRCTL whereas the real-time nature allows us to form the basis for this by providing numbers of exponential size.

Definition 5.2. Let $\mathfrak{A}$ be a class of timed automata, let $k \geq 1$, and let *ops* be a set of (temporal or interval) operators like $\mathtt{EF}$, $=, \leq$ etc. An $(\mathfrak{A}, k, ops)$*-encoding of large numbers* is a sequence $Enc = (\mathcal{A}_n, \langle\cdot\rangle_n, zero_n, max_n, inc_n, dec_n, eq_n^0)_{n\geq 0}$ such that, for all $n \geq 0$, the following hold:

- $\mathcal{A}_n$ is a timed automaton from $\mathfrak{A}$ of polynomial size in n over k clocks,
- $\langle\cdot\rangle_n$ is a function that assigns a set of states $\langle m\rangle_n$ in the timed transition system $\mathcal{T}_{\mathcal{A}_n}$ to any $m \in [2^{2^n}]$,
- $zero_n$ and max_n are closed formulas of $\mathrm{TRCTL}_k[ops]$ satisfying

$$[\![zero_n]\!]^{\mathcal{A}_n} = \langle 0\rangle_n \ , \qquad [\![max_n]\!]^{\mathcal{A}_n} = \langle 2^{2^n} - 1\rangle_n$$

- $inc_n(x), dec_n(x)$ are formulas of $\mathrm{TRCTL}_k[ops]$, each with a single propositional variable x, satisfying for all $m \in [2^{2^n}]$:

$$[\![inc_n(x)]\!]^{\mathcal{A}_n}_{[x\mapsto\langle m\rangle_n]} = \langle m+1 \mod 2^{2^n}\rangle_n \ ,$$
$$[\![dec_n(x)]\!]^{\mathcal{A}_n}_{[x\mapsto\langle m\rangle_n]} = \langle m-1 \mod 2^{2^n}\rangle_n \ ,$$

- $eq_n^0(x)$ is a formula of $\mathrm{TRCTL}_k[ops]$, with a single propositional variable x, satisfying for all $m \in [2^{2^n}]$:

$$s_0 \in [\![eq_n^0(x)]\!]^{\mathcal{A}_n}_{[x\mapsto\langle m\rangle_n]} \text{ iff } m = 0$$

  where s_0 is the initial state of $\mathcal{T}_{\mathcal{A}_n}$.

Additionally, all these formulas need to be of polynomial size in n and be defined over the same k clocks as the $\mathcal{A}_n$.

In other words, a $(\mathfrak{A}, k, ops)$-encoding of large numbers provides a way to represent numbers up to doubly exponential size in n for any given $n \geq 0$, defining 0 and $2^{2^n} - 1$ and to increase, decrease and test them for being equal to 0 using formulas of $\mathrm{TRCTL}_k[ops]$.

The condition on eq_n^0 might seem counter-intuitive at first, given its restriction to the starting state of $\mathcal{T}_{\mathcal{A}_n}$ instead of a constraint applicable at all states in $\mathcal{T}_{\mathcal{A}_n}$. However, as we will see shortly, this is both sufficient and necessary: it is sufficient since the generic hardness proof mostly "happens at" the initial state in the sense that the encoding happens in the *arguments* of the corresponding formula, which are sets of pairs of a location and a clock evaluation. Hence, the formula does not manipulate time in the classic sense of using $\mathtt{EU}$ etc., but rather it manipulates sets of the aforementioned form. Restricting the condition to the initial state is necessary as the manipulation of such arguments happens at an abstract level where sets or states are transformed and, hence, referring to the actual clock value is not possible.

We will often drop the index $\cdot_n$ in the components of such an encoding and simply write $\langle m\rangle$ instead of $\langle m\rangle_n$, *inc* instead of inc_n and so on when n can be derived from the context.

Given an encoding of large numbers we can define further formulas as abbreviations, namely

$$eq_n^1(x) := eq_n^0(dec_n(x)) \qquad gt_n^0(x) := \neg eq_n^0(x) \qquad lt_n^{\max}(x) := \neg eq_n^0(inc_n(x))$$
$$eq_n^{\max}(x) := eq_n^0(inc_n(x)) \qquad gt_n^1(x) := gt_n^0(x) \wedge \neg eq_n^1(x)$$

The following is an immediate consequence of the properties demanded in Definition 5.2 and the semantics of the Boolean operators.

Lemma 5.3. *The following hold for any given n in any given encoding Enc:*

$$\begin{aligned} s_0 &\in [\![eq_n^1(x)]\!]^{\mathcal{A}_n}_{[x\mapsto\langle m\rangle_n]} \text{ iff } m = 1,\\ s_0 &\in [\![eq_n^{\max}(x)]\!]^{\mathcal{A}_n}_{[x\mapsto\langle m\rangle_n]} \text{ iff } m = 2^{2^n}-1,\\ s_0 &\in [\![gt_n^0(x)]\!]^{\mathcal{A}_n}_{[x\mapsto\langle m\rangle_n]} \text{ iff } m > 0,\\ s_0 &\in [\![gt_n^1(x)]\!]^{\mathcal{A}_n}_{[x\mapsto\langle m\rangle_n]} \text{ iff } m > 1,\\ s_0 &\in [\![lt_n^{\max}(x)]\!]^{\mathcal{A}_n}_{[x\mapsto\langle m\rangle_n]} \text{ iff } m < 2^{2^n}-1. \end{aligned}$$

The existence of such encodings suffices to prove 2-EXPTIME-hardness of model checking for corresponding TRCTL-fragments over corresponding classes of timed automata.

Theorem 5.4. *If there is an $(\mathfrak{A}, k, ops)$-encoding of large numbers Enc, then the model checking problem for* TRCTL$_k[ops]$ *over the class $\mathfrak{A}$ of timed automata is* 2-EXPTIME*-hard.*

Proof. Let $Enc = (\mathcal{A}_n, \langle\cdot\rangle_n, zero_n, max_n, inc_n, dec_n, eq_n^0)_{n\geq 0}$ be such an encoding, and suppose that a DTM $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta, q_{\text{acc}})$ and some $n \in \mathbb{N}$ are given. We need to construct a timed automaton $\mathcal{A}_{\mathcal{M},n}$ and a TRCTL$_k[ops]$-formula $\varphi_{\mathcal{M},n}$ such that there is a 2^{2^n}-certificate for $\mathcal{M}$ and n iff $\mathcal{T} \models \varphi_{\mathcal{M},n}$ for the timed transition system $\mathcal{T}$ that arises from the timed automaton $\mathcal{A}_{\mathcal{M},n}$. We simply let $\mathcal{A}_{\mathcal{M},n} := \mathcal{A}_n$ that is given by *Enc*.

For the construction of $\varphi_{\mathcal{M},n}$ let $\hat{\Gamma} = \{a_1, \ldots, a_m\}$ and $\hat{\delta}$ be as defined in Sect. 5.1, resulting from Q, Γ, δ. Then we define $\varphi_{\mathcal{M},n}$ as

$$\texttt{rec}_{(q_{\text{acc}},\Box)} \left(\begin{array}{c} \vdots \\ \mathcal{C}_{a_i}(t,s) \;.\; chk_{a_i}(t,s) \;\vee \bigvee\limits_{(b_1,b_2,b_3,a_i)\in\hat{\delta}} nxt_{b_1,b_2,b_3}(t,s) \\ \vdots \end{array} \right)(max, zero)$$

where

$$chk_a(t,s) := \begin{cases} eq^0(s) \vee eq_n^{\max}(s) & , \text{ if } a = \# \\ eq^0(t) \wedge eq_n^1(s) & , \text{ if } a = (q_0, \Box) \\ gt^1(s) \wedge lt_n^{\max}(s) & , \text{ if } a = \Box \\ \texttt{ff} & , \text{ otherwise} \end{cases}$$

$$\begin{aligned} nxt_{b_1,b_2,b_3}(t,s) := {} & gt^0(t) \wedge gt^0(s) \wedge lt^{\max}(s) \wedge \\ & \mathcal{C}_{b_1}(dec(t), dec(s)) \wedge \mathcal{C}_{b_2}(dec(t), s) \wedge \mathcal{C}_{b_3}(dec(t), inc(s)). \end{aligned}$$

Clearly, $\mathcal{A}_n$ only uses k clocks by definition and is of polynomial size in $|\mathcal{M}|$ and n. Moreover, $\varphi_{\mathcal{M},n}$ does not use more clocks than the k given ones either (which may occur in the subformulas inc_n etc.) or any other temporal or interval operators than those defined in *ops* already, and it is of such polynomial size, too. Hence, this construction yields a polynomially-sized instance of the model checking problem for the TRCTL$_k[ops]$-fragment.

We now claim that $\mathcal{T}_{\mathcal{A}_n} \models \varphi_{\mathcal{M},n}$ iff there is a 2^{2^n}-certificate for $\mathcal{M}$. This follows from the fact that the definition of the $\mathcal{C}_{a_i}$ mirrors the pattern of the certificate *Cert* described in Sec. 5.1, and correctness of the arithmetic follows from Definition 5.2 and Lemma 5.3. Note that $\varphi_{\mathcal{M},n}$ is well-defined w.r.t. monotonocity of the $\mathcal{C}_{a_i}$ since all of them occur only positively in $nxt_{b_1,b_2,b_3}(t,s)$. The variables s and t occur both positively and negatively but they are not recursion variables, so this is unproblematic. □

Here we see why the conditions on eq_n^0, eq_n^1, etc. were sufficient: The functions inc_n and dec_n,which manipulate sets, always appear in an operand position, while eq_n^0 etc. are always used after no time has actually elapsed and no transitions have happened, whence it is sufficient that they be well-defined on the initial state of the TLTS in question.

Note that the input to a model checking problem is a pair consisting of a system description and a formalisation of a correctness property. The complexity is measured in the size of the pair, but the two parts play very different roles under different view points. There are two established metrics for measuring the model checking complexity of a logic in each parameter separately, the data complexity and the expression complexity. The *data complexity* is the complexity of model checking for any fixed formula, measured in the size of the system description. It is an important measure in program specification where correctness properties are often small and vary little while systems descriptions are much larger and vary a lot more. The *expression complexity* is the complexity of model checking for any fixed transition system. It is an important measure of the expressive power of the logic. The result above in Theorem 5.4 can be refined show that the model checking problem for TRCTL$_k[ops]$ is 2-EXPTIME-hard already in its expression complexity for many instances of *ops*.

Corollary 5.5. *Suppose there is a* $(\mathfrak{A}, k, ops)$*-encoding of large numbers Enc defining a constant sequence of timed automata* $(\mathcal{A}_n)_{n\geq 0}$*, i.e.* $\mathcal{A}_n = \mathcal{A}_m$ *for all* n, m*. Then the expression complexity of* $\mathrm{TRCTL}_k[ops]$ *over the class* $\mathfrak{A}$ *of timed automata is already* 2-EXPTIME*-hard.*

Proof. This follows immediately from the observation that, in this case, the construction in the proof of Theorem 5.4 yields a fixed timed automaton independent of the DTM $\mathcal{M}$ and the parameter n. □

Remark 5.6. In the following, we will establish hardness in the sense of the Corollary 5.5 for various fragments of TRCTL_1, i.e. for formulas and TA using only one clock. On the other hand, we have chosen to introduce the definition of $(\mathfrak{A}, k, ops)$-encodings of large numbers, and Theorem 5.4 in a form that is parameterised in the number of clocks. This was done to keep the result as general as possible, and to leave open further work that might produce hardness results using even less expressive fragments, which then might need to revert to more than one clock. Such fragments exist, for example, when restricting expressive power below that of TCTL, see e.g. [19].

5.3. Hardness proofs for several fragments

Our goal is now to show 2-EXPTIME-hardness for several fragments of TRCTL by showing that they satisfy the requirements of the generic hardness proof given in Theorem 5.4, and even those for the expression complexity given in Corollary 5.5. As a minimum, these fragments will contain the temporal operator $\mathtt{EF}$ and one of the one of the three sets of clock constraints $\{=\}, \{<, >\}, \{\leq, \geq\}$ or $\{[d, d']\}$ for any fixed $d \leq d'$ as permitted clock constraints for $\mathtt{EF}$. Except in the last case, propositional clock constraints are also restricted to use only $=$ or $<, >$ or $\leq, \geq$, while in the last case, we need propositional clock constraints of arbitrary forms. The first set is contained in the last one for $d = d' = 1$, but we will see that it makes sense to prove 2-EXPTIME-hardness for both of them separately since the first set is more accessible and has stronger restrictions on clock constraints.

Recall that Theorem 5.4 requires us to provide a $(\mathfrak{A}, k, ops)$-encoding of large numbers for each of the four sets mentioned above. Hence, we have to provide a sequence $(\mathcal{A}_n, \langle\cdot\rangle_n, zero_n, max_n, inc_n, dec_n, eq^0_n)_{n\geq 0}$ that satisfies the requirements of the theorem, namely that $\langle\cdot\rangle_n$ assigns a unique set of states in $\mathcal{T}_{\mathcal{A}_n}$ to each number in $[2^{2^n}]$, which can then be queried or manipulated using $zero_n, max_n, inc_n$ and dec_n. It is immediate that this requires $\mathcal{T}_{\mathcal{A}_n}$ to have at least exponentially many states in n, yet $\mathcal{A}_n$ is required to have size polynomial in n. Hence, there is no hope to achieve this using simply exponentially many different locations. This is also not surprising, since not using at least one clock would make the whole problem collapse to the RecCTL model checking problem, which is just EXPTIME-complete. Somewhat surprisingly, we can actually make do with a one-state timed automaton that contains no propositions, no transitions and has only one clock $\mathtt{z}$. Note that this immediately satisfies the requirements of Corollary 5.5 provided that we can satisfy those of Theorem 5.4.

Given that we have resorted to a one-state automaton, the whole mechanics of encoding large numbers will have to happen through clock values and clock constraints. Hence, from now on we will identify clock values and states, since the location component in a state is always the same. Since we want to produce polynomially-sized formulas, simply using clock constraints directly to implement the formulas required by Theorem 5.4 will not work even for a binary encoding of numbers. However, we can, in fact, encode and manipulate numbers, or rather their binary representations, using constantly many formulas. Recall how binary incrementation and decrementation works: A bit is set in the binary representation of $m + 1$ if either

- it is set in the representation of m, and a bit of lesser significance is not set there, or
- it is not set in the representation of m, but all bits of lesser significance are set there.

Decrementation uses a similar pattern. It follows that, in order to decide whether a bit is set in the binary representation of the increment or decrementat of a number, it suffices to know the values of all bits of lesser significance in its representation, as well as the value of the bit itself. In fact, it is enough to know whether some bit of lower significance is set or not set, and the value of the bit itself. Theorem 5.4 requires us to assign a specific set of states in the system generated by our one-state automaton to each number in $[2^{2^n}]$, i.e. a set of clock values. Hence, it is enough to find 2^n many different clock values that play the roles of the bits in the encoding of large numbers. A bit is set in the set $\langle m\rangle_n$ iff the respective clock value is contained in this set.

There are two difficulties with this approach: The operators $\mathtt{EF}$ and $\mathtt{AG}$, i.e. the passage of time, only allow controlled *increases* of $\mathtt{z}$. Hence, we decide that bits of lesser significance shall be encoded by larger clock values, while smaller clock values encode bits of larger significance. Moreover, the operators $\mathtt{EF}$ and $\mathtt{AG}$, unless properly controlled via clock constraints, always talk about the uncountably many different clock values that are reachable via the passage of time, while we only have finitely many bits that need to be represented. Hence, we have to be quite careful when designing our formulas, inc_n, etc. The general idea is to make sure that sets are manipulated such that the set of all clock values can be partitioned into finitely many intervals that are easy to control.

In the following, let $\mathcal{A}$ be the TA $(\{\ell\}, \{\mathtt{z}\}, \ell, \iota, \emptyset, \lambda)$ with $\iota(\ell) = \top$ and $\lambda(\ell) = \emptyset$, i.e. the one-state automaton with no propositions, transitions or clock constraints. Let $\mathcal{S} = \{(\ell, r) \mid r \geq 0\}$ be its state space.

The case of ops $= \{\mathtt{EF}, \leq, \geq\}$ We begin with the case where the only available clock constraints, both as atomic propositions and also when used as the subscript of $\mathtt{EF}$, are of the form $\mathtt{z} \leq k$ and $\mathtt{z} \geq k$ for arbitrary k. Note that clock constraints of the form $\mathtt{z} = k$, $\mathtt{z} > k$ and $\mathtt{z} < k$ are also available via the obvious boolean combinations, but only as propositions, not as interval bounds.

Our bits will be half-open intervals of clock values of the form $[k, k+1)$ and we will make sure that we only ever have to consider sets that either contain all clock values from such an interval, or none. Given such a set $x \subseteq \mathcal{S}$, it encodes the value $m = \sum_{0 \leq k \leq 2^n-1} b_k \cdot 2^k$ where $b_k = 1$ if $(\ell, 2^n - 1 - k) \in x$ and $b_k = 0$ otherwise. Note that the non-integral clock values do not appear in this definition, and that the most significant bit is that where the clock has value 0. In other words, (ℓ, k) is to be included in the representation of m iff the $k+1$st most significant bit is set in the standard binary representation of m, i.e. iff $\lfloor \frac{m}{2^{2^n-1-k}} \rfloor \equiv 1 \mod 2$.

Lemma 5.7. *The sequence* $(\mathcal{A}_n, \langle\cdot\rangle_n, zero_n, max_n, inc_n, dec_n, eq^0_n)_n$ *defined via*

- $\mathcal{A}_n = \mathcal{A}$,
- $\langle m \rangle_n := \bigcup_{0 \leq k \leq 2^n-1} \{(\ell, r) \mid \lfloor \frac{m}{2^{2^n-1-k}} \rfloor \equiv 1 \mod 2$ *and* $k \leq r < k+1\}$,
- $zero_n := \mathtt{ff}$ *and* $max_n := \mathtt{z} < 2^n - 1$,
- $inc_n(x) := max_n \wedge \big((x \wedge \mathtt{EF}_{\geq 1}(\neg x \wedge max_n) \vee (\neg x \wedge \mathtt{AG}_{\geq 1}(max_n \to x))\big)$,
- $dec_n(x) := max_n \wedge \big((x \wedge \mathtt{EF}_{\geq 1}(x \wedge max_n) \vee (\neg x \wedge \mathtt{AG}_{\geq 1}(max_n \to \neg x))\big)$,
- $eq^0_n(x) := \neg x \wedge \mathtt{AG}_{\geq 1} \neg x$

is an $(\{\mathcal{A}\}, 1, \{\mathtt{EF}, \leq, \geq\})$*-encoding of large numbers.*

Proof. Obviously, the formulas above satisfy all the syntactic requirements for encodings of large numbers. In particular, since they use only constantly many clock constraints for numbers of at most exponential size, the size of the formulas is indeed polynomial in n if those numbers are given in binary. Hence, we focus on the semantic aspects of Definition 5.2.

Recall that the definition of $\langle\cdot\rangle_n$ partitions the interval $[0, 2^n - 1)$ of clock values into 2^n many half-open intervals of the form $[k, k+1)$, and that these are the bits we use in the encoding, with $[0, 1)$ encoding the most significant bit and $[2^n - 2, 2^n - 1)$ encoding the least significant one. The crucial part here is that, for each interval $[k, k+1)$, the encoding of a large number either contains all states (ℓ, r) with $r \in [k, k+1)$ or none of them. Clearly this is the case for $zero_n$ and max_n, as these define either the empty set or the set $\{(\ell, r) \mid 0 \leq r < 2^n - 1\}$. The encoding $\langle m \rangle_n$ then formalises the correspondence between bits in the binary representation of m and intervals whose clock values are contained in $\langle m \rangle_n$: For all $0 \leq k < 2^n - 1$, the kth bit, starting from the least significant one, is set in the binary representation of m iff $\langle m \rangle_n$ contains all states of the form (ℓ, r) with $r \in [2^n - 2 - k, 2^n - 1 - k)$.

The requirements of Definition 5.2 on eq^0_n, $zero_n$ and max_n are straightforward. Hence, it remains to show that inc_n and dec_n both maintain the partition into intervals of the form $[k, k+1)$ and that they indeed encode binary incrementation and decrementation, respectively. Towards the former, note that if the semantics of x is a union of sets of the form $\{(\ell, r) \mid k \leq r < k+1\}$, then so is the semantics of $\mathtt{EF}_{\geq 1} x$ and that of derived formulas: If k is the biggest number such that the set $\{(\ell, r) \mid k \leq r < k+1\}$ is in the semantics of x, then $(\ell, k - \epsilon)$ is in the semantics of $\mathtt{EF}_{\geq 1} x$ for all $\epsilon \geq 0$, but not (ℓ, k) itself. The claims w.r.t. binary incrementation and decrementation are then shown via straightforward verification. □

The case of ops $= \{\mathtt{EF}, <, >\}$ This fragment is quite similar to the previous one.

Lemma 5.8. *The sequence* $(\mathcal{A}_n, \langle\cdot\rangle_n, zero_n, max_n, inc_n, dec_n, eq^0_n)_n$ *defined via*

- $\mathcal{A}_n = \mathcal{A}$,
- $\langle m \rangle_n := \bigcup_{0 \leq k \leq 2^n-1} \{(\ell, r) \mid \lfloor \frac{m}{2^{2^n-1-k}} \rfloor \equiv 1 \mod 2$ *and* $k \leq r < k+1\}$,
- $zero_n := \mathtt{ff}$ *and* $max_n := \mathtt{z} < 2^n - 1$,
- $inc_n(x) := max_n \wedge \big((x \wedge \mathtt{EF}_{>1}(\neg x \wedge max_n \vee (\neg x \wedge \mathtt{AG}_{>1}(max_n \to x))\big)$,
- $dec_n(x) := max_n \wedge \big((x \wedge \mathtt{EF}_{>1}(x \wedge max_n) \vee (\neg x \wedge \mathtt{AG}_{>1}(\neg max_n \to \neg x))\big)$,
- $eq^0_n(x) := \neg x \wedge \mathtt{AG}_{>1} \neg x$

is an $(\{\mathcal{A}\}, 1, \{\mathtt{EF}, <, >\})$*-encoding of large numbers.*

Proof. The proof is very similar to that of Lemma 5.7. The syntactic difference is that the definition of max_n changes, and that in the clock constraints at $\mathtt{EF}$ and $\mathtt{AG}$ the condition ≥ 1 has been replaced by > 1. However, note that due to the way that half-open intervals work, there is no semantic difference here: If the semantics of x is a (union of) half-open interval(s) of the form $\{(\ell, r) \mid k \leq r < k+1\}$ and (ℓ, r') is in the semantics of $\mathtt{EF}_{\geq 1} x$, then there is some r' and $i \geq 1$ such that $r' + i \in [k, k+1)$. In particular, $r' + i < k+1$, so there is $\epsilon > 0$ such that $r' + i + \epsilon < k+1$, too. □

The case of ops $= \{\mathtt{EF}, =\}$ If we restrict ourselves to equalities in the clock constraints, the constructions from the previous paragraph do not transfer without some adjustments. In particular, is not clear how a constraint of the form $\mathtt{z} < 2^n - 1$ can be replaced, and how the half-open intervals from the previous paragraph can be addressed, given that time can only flow in integral units.

Towards the former, let $\mathtt{EF}^*(\varphi)$ be a macro defined as $\mathtt{rec}\, \mathcal{F}\ \phi \vee \mathtt{EF}_{=1}\mathcal{F}$ and $\mathtt{AG}^*(\varphi)$ as $\neg\mathtt{EF}^*(\neg\varphi)$. Clearly, $\mathtt{EF}^*\mathtt{z} = 2^n - 2$ entails $\mathtt{z} < 2^n - 1$ on *integral* clock values. Note that the initial state of any timed LTS has clock value 0 for all clocks, and if time is only allowed to flow in integral units, all relevant clock values towards the semantics of a formula will only depend on states with integral clock values. Hence, instead of half-open intervals of the form $[k, k+1)$, we use point intervals of the form $[k, k]$.

Lemma 5.9. *The sequence* $(\mathcal{A}_n, \langle\cdot\rangle_n, zero_n, max_n, inc_n, dec_n, eq_n^0)_n$ *defined via*

- $\mathcal{A}_n = \mathcal{A}$,
- $\langle m\rangle_n := \bigcup_{0\le k\le 2^n-1}\{(\ell, k) \mid \lfloor \frac{m}{2^{2^n-1-k}} \rfloor \equiv 1 \mod 2\}$,
- $zero_n := \mathtt{ff}$ *and* $max_n := \mathtt{EF}^*\mathtt{z} = 2^n - 2$,
- $inc_n(x) := max_n \wedge \big((x \wedge \mathtt{EF}^*(\neg x \wedge max_n) \vee (\neg x \wedge \mathtt{AG}^*(max_n \to x)\big)$,
- $dec_n(x) := max_n \wedge \big((x \wedge \mathtt{EF}^*(x \wedge max_n) \vee (\neg x \wedge \mathtt{AG}^*(max_n \to \neg x)\big)$,
- $eq_n^0(x) := \mathtt{AG}^*\neg x$

is an $(\{\mathcal{A}\}, 1, \{\mathtt{EF}, =\})$*-encoding of large numbers.*

Proof. By verification of the claims on $\mathtt{EF}^*$ and $\mathtt{AG}^*$. □

The case of ops $= \{\mathtt{EF}, [d, d']^*\}$ Finally, we study the case where the available clock constraints contain at least one finite interval of the form $[d, d']$ with $d \le d'$ natural numbers. The interesting part here are the clock constraints used in conjunction with the temporal operators. The previous proof pattern still works, even if temporal operators are restricted to ones of the form of e.g. $\mathtt{EF}_{[3,5]}$, i.e. if one can only make statements on the flow of time in increments somewhere between at least 3 and at most 5 units. The crucial observation is that only the *lower bound* of the interval matters; it mostly suffices to adapt to such a lower bound by stretching the area on which encodings happen by this factor. Hence, our bits will be half-open intervals of the form $[d \cdot k, d \cdot (k+1))$.

However, as a divergence from the previous cases, we have to use a clock constraint of a different form than $\mathtt{z} \in [d, d']$ in order to define max_n. Hence, for *propositional* constraints we allow ourselves to use arbitrary clock constraints, but interval bounds on *temporal quantifiers* are restricted to one fixed interval $[d, d']$. By writing $[d, d']^*$ instead of $[d, d']$, we signal that the restriction to $[d, d']$ is valid only for clock constraints at temporal operators, while propositional clock constraints of other forms are also used.

Lemma 5.10. *Let* $1 \le d \le d'$ *be integers. Let* $\mathtt{EF}^*\varphi := \mathtt{rec}\, \mathcal{F}\ \varphi \vee \mathtt{EF}_{[d,d']}\mathcal{F}$ *and* $\mathtt{AG}^*\varphi := \neg\mathtt{EF}^*\neg\varphi$.
The sequence $(\mathcal{A}_n, \langle\cdot\rangle_n, zero_n, max_n, inc_n, dec_n, eq_n^0)_n$ *defined via*

- $\mathcal{A}_n = \mathcal{A}$,
- $\langle m\rangle_n := \bigcup_{0\le k\le 2^n-1}\{(\ell, r) \mid \lfloor \frac{m}{2^{2^n-1-k}} \rfloor \equiv 1 \mod 2$ *and* $d \cdot k \le r < d \cdot (k+1)\}$,
- $zero_n := \mathtt{ff}$ *and* $max_n := \mathtt{z} \in [0, d \cdot 2^n - 2]$,
- $inc_n(x) := max_n \wedge \big((x \wedge \mathtt{EF}^*(\neg x \wedge max_n)) \vee (\neg x \wedge \mathtt{AG}^*(max_n \to x))\big)$,
- $dec_n(x) := max_n \wedge \big((x \wedge \mathtt{EF}^*(x \wedge max_n) \vee (\neg x \wedge \mathtt{AG}^*(max_n \to \neg x)\big)$,
- $eq_n^0(x) := \neg x \wedge \mathtt{AG}^*\neg x$

is an $(\{\mathcal{A}\}, 1, \{\mathtt{EF}, [d, d']^*\})$*-encoding of large numbers.*

Proof. By verification. □

Remark 5.11. The above result also holds if the interval bounds are equal, i.e. for point intervals, in which case it collapses to the previous result. It also holds for open and half-open intervals. The extension to open or half-open interval bounds follows a similar pattern as the extension from $\le$ and $\ge$ to $<$ and $>$ does.

Putting it all together From the above, and Theorem 4.3, we obtain that the TRCTL model checking problem is 2-EXPTIME-hard already in very restricted settings.

Theorem 5.12. *The* TRCTL *model checking-problem is* 2-EXPTIME*-hard already in expression complexity for all fragments that contain at least the operators* EF *and clock constraints of the form* $\ge, >, =$ *already over TA with one clock only. Moreover, let* $1 \le d \le d'$ *be*

natural numbers. Then, for fragments that contain at least $\mathtt{EF}_{[d,d']}$ *and arbitrary propositional clock constraints, the hardness result also holds, and similarly for open and half-open intervals.*

This results is to be contrasted with similar results for TCTL where there are clear complexity differences between the settings for one clock, two clocks, or more than two clocks, and where also the kind of clock constraints available matters [19]. The explanation for this is that the power added through recursion is so big that it completely overshadows all the subtle differences that arise in the non-recursive setting, since e.g. iterated reachability by multiples of a time unit (cf. $\mathtt{EF}^*$ above) can be expressed.

Remark 5.13. To our knowledge, the hardness in expression complexity is lost if formulas of the form $z \leq 2^n - 1$ etc. are not available. However, the general hardness results, i.e. those for the combined complexity, persist even in the one-clock setting; one simply has to modify the underlying TA to be parameterised in n to enforce the necessary conditions via location constraints. Since having these formulas present adds no complexity to the model checking problem, keeping them in the logic appears natural.

6. Conclusion & further work

We have introduced Timed Recursive Temporal Logic (TRCTL) and shown that its model checking problem is 2-EXPTIME-complete, already over TLTS generated by TA with one clock only and with very restricted temporal operators and clock constraints available. It should be noted that these lower bounds contrast a much richer complexity landscape present in TCTL w.r.t. the number of clocks or the clock constraints available [19].

TRCTL's satisfiability problem is undecidable; this is inherited from Recursive Temporal Logic [13]. TRCTL is strictly stronger in expressive power than its two constituent parts RecCTL and TCTL since either can express properties that the other cannot, namely higher-order properties [13] or real-time properties. A fine-grained comparison of the expressive power TRCTL against that of TCTL is still to be done, i.e. it is open exactly which properties can be expressed in TRCTL, but not in TCTL.

Further research concerns two angles: practicability and extensions in expressive power. With respect to the former, the 2-EXPTIME-complete model checking problem might seem prohibitive, yet higher-order algorithms are open to optimisations that can yield surprisingly competitive algorithms [27,28]. The latter angle includes straightforward extensions such as propositions that test for the value of some clock that are unlikely to require new methods, but also more intricate ones like diagonal constraints etc. which, of course, are also likely to lead to undecidability [29].

CRediT authorship contribution statement

Florian Bruse: Writing – review & editing, Writing – original draft. **Martin Lange:** Writing – review & editing, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] A. Pnueli, The temporal logic of programs, in: Proc. 18th Symp. on Foundations of Computer Science, FOCS'77, IEEE, Providence, RI, USA, 1977, pp. 46–57, https://doi.org/10.1109/SFCS.1977.32.
[2] E.A. Emerson, E.M. Clarke, Using branching time temporal logic to synthesize synchronization skeletons, Sci. Comput. Program. 2 (3) (1982) 241–266, https://doi.org/10.1016/0167-6423(83)90017-5.
[3] E.A. Emerson, J.Y. Halpern, "Sometimes" and "not never" revisited: on branching versus linear time temporal logic, J. ACM 33 (1) (1986) 151–178.
[4] E.A. Emerson, C.S. Jutla, The complexity of tree automata and logics of programs, SIAM J. Comput. 29 (1) (2000) 132–158, https://doi.org/10.1137/s0097539793304741.
[5] M.Y. Vardi, Why is modal logic so robustly decidable?, in: Proc. DIMACS Workshop on Descriptive Complexity and Finite Models, in: DIMACS Series in Discr. Math. and Theor. Comp. Sci., vol. 31, DIMACS/AMS, 1996, pp. 149–183.
[6] A.P. Sistla, E.M. Clarke, N. Francez, A.R. Meyer, Can message buffers be axiomatized in linear temporal logic?, Inf. Control 63 (1/2) (1984) 88–112.
[7] M. Viswanathan, R. Viswanathan, A higher order modal fixed point logic, in: CONCUR'04, in: LNCS, vol. 3170, Springer, 2004, pp. 512–528, https://doi.org/10.1007/978-3-540-28644-8_33.
[8] D. Harel, A. Pnueli, J. Stavi, Propositional dynamic logic of nonregular programs, J. Comput. Syst. Sci. 26 (2) (1983) 222–243, https://doi.org/10.1016/0022-0000(83)90014-4.
[9] M. Müller-Olm, A modal fixpoint logic with chop, in: Proc. 16th Symp. on Theoretical Aspects of Computer Science, STACS'99, in: LNCS, vol. 1563, Springer, 1999, pp. 510–520, https://doi.org/10.1007/3-540-49116-3_48.

[10] M. Lange, Model checking propositional dynamic logic with all extras, J. Appl. Log. 4 (1) (2005) 39–49, https://doi.org/10.1016/j.jal.2005.08.002.
[11] M. Lange, C. Stirling, Model checking fixed point logic with chop, in: Proc. 5th Conf. on Foundations of Software Science and Computation Structures, FOSSACS'02, in: LNCS, vol. 2303, Springer, 2002, pp. 250–263, https://doi.org/10.1007/3-540-45931-6_18.
[12] R. Axelsson, M. Lange, R. Somla, The complexity of model checking higher-order fixpoint logic, Log. Methods Comput. Sci. 3 (2007) 1–33, https://doi.org/10.2168/LMCS-3(2:7)2007.
[13] F. Bruse, M. Lange, Temporal logic with recursion, in: Proc. 27th Int. Symp. on Temporal Representation and Reasoning, TIME'20, in: LIPIcs, vol. 178, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 6:1–6:14, https://doi.org/10.4230/LIPIcs.TIME.2020.6.
[14] D. Kozen, Results on the propositional μ-calculus, Theor. Comput. Sci. 27 (1983) 333–354, https://doi.org/10.1016/0304-3975(82)90125-6.
[15] D. Janin, I. Walukiewicz, On the expressive completeness of the propositional μ-calculus with respect to monadic second order logic, in: Proc. 7th Conf. on Concurrency Theory, CONCUR'96, in: LNCS, vol. 1119, Springer, 1996, pp. 263–277.
[16] R. Alur, T.A. Henzinger, Logics and models of real time: a survey, in: Real-Time: Theory in Practice, REX'91 Workshop, in: LNCS, vol. 600, Springer, 1992, pp. 74–106.
[17] R. Alur, D.L. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (2) (1994) 183–235, https://doi.org/10.1016/0304-3975(94)90010-8.
[18] R. Alur, C. Courcoubetis, D. Dill, Model-checking in dense real-time, Inf. Comput. 104 (1) (1993) 2–34, https://doi.org/10.1006/inco.1993.1024.
[19] F. Laroussinie, N. Markey, P. Schnoebelen, Model checking timed automata with one or two clocks, in: Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04, in: LNCS, vol. 3170, Springer, 2004, pp. 387–401, https://doi.org/10.1007/978-3-540-28644-8_25.
[20] A.K. Chandra, D. Kozen, L.J. Stockmeyer, Alternation, J. ACM 28 (1) (1981) 114–133, https://doi.org/10.1145/322234.322243.
[21] F. Bruse, M. Lange, Model checking timed recursive CTL, in: Proc. 28th Int. Symp. on Temporal Representation and Reasoning, TIME'21, in: LIPIcs, vol. 206, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 12:1–12:14.
[22] C. Baier, J.-P. Katoen, Principles of Model Checking, MIT Press, 2008.
[23] R. Alur, C. Courcoubetis, D. Dill, Model-checking for real-time systems, in: Proc. 5th Ann. IEEE Symp. on Logic in Computer Science, LICS'90, IEEE Computer Society Press, 1990, pp. 414–427, https://doi.org/10.1109/LICS.1990.113766.
[24] H. Bekić, Programming Languages and Their Definition, Selected Papers, LNCS, vol. 177, Springer, 1984.
[25] A. Arnold, D. Niwiński, Rudiments of μ-Calculus, Studies in Logic and the Foundations of Mathematics, vol. 146, North-Holland, 2001.
[26] A. Tarski, A lattice-theoretical fixpoint theorem and its application, Pac. J. Math. 5 (1955) 285–309, https://doi.org/10.2140/pjm.1955.5.285.
[27] F. Bruse, J. Kreiker, M. Lange, M. Sälzer, Local higher-order fixpoint iteration, in: Proc. 11th Int. Symp. on Games, Automata, Logics, and Formal Verification, GandALF'20, in: EPTCS, vol. 326, 2020, pp. 97–113, https://doi.org/10.4204/EPTCS.326.7.
[28] Y. Hosoi, N. Kobayashi, T. Tsukada, A type-based HFL model checking algorithm, in: Proc. 17th Asian Symp. on Programming Languages and Systems, APLAS'19, in: NCS, vol. 11893, Springer, 2019, pp. 136–155, https://doi.org/10.1007/978-3-030-34175-6_8.
[29] P. Bouyer, F. Laroussinie, N. Markey, J. Ouaknine, J. Worrell, Timed temporal logics, in: Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday, in: LNCS, vol. 10460, Springer, 2017, pp. 211–230, https://doi.org/10.1007/978-3-319-63121-9_11.