The Tail-Recursive Fragment of Timed Recursive CTL

Florian Bruse
School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Martin Lange
School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Etienne Lozes
Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis,
Université Côte d'Azur, France

Abstract

Timed Recursive CTL (TRCTL) was recently proposed as a merger of two extensions of the well-known branching-time logic CTL: Timed CTL on one hand is interpreted over real-time systems like timed automata, and Recursive CTL (RecCTL) on the other hand obtains high expressiveness through the introduction of a recursion operator. Model checking for the resulting logic is known to be 2-EXPTIME-complete.

The aim of this paper is to investigate the possibility to obtain a fragment of lower complexity without losing too much expressive power. It is obtained by a syntactic property called "tail-recursiveness" that restricts the way that recursive formulas can be built. This restriction is known to decrease the complexity of model checking by half an exponential in the untimed setting. We show that this also works in the real-time world: model checking for the tail-recursive fragment of TRCTL is EXPSPACE-complete. The upper bound is obtained by a standard untiming construction via region graphs, and rests on the known complexity of tail-recursive fragments of higher-order modal logics. The lower bound is established by a reduction from a suitable tiling problem.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Program specifications

Keywords and phrases formal specification, temporal logic, real-time systems

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.5

1 Introduction

Models of systems that incorporate real-time aspects play an important role in the specification and verification of the behaviour of embedded systems. Correct functioning of such systems often depends on the satisfaction of constraints that involve concrete times like "*the wing flaps are adjusted within 5msec of a change in vertical angle reported by the gyrometer sensor.*"

Timed automata [3] are a standard model for the abstraction of the behaviour of real-time systems which has been studied well, including ways to extend their expressiveness, cf. [4, 7]. The desired behaviour of such dynamic systems is typically specified using temporal logics that formalise statements about the evolution of such a system's behaviour in time. For example, the above property in a formal syntax yields a formula like $\mathtt{AG}(\mathsf{chng} \to \mathtt{AF}_{\leq 5}\mathsf{adj})$.

This formula belongs to the real-time temporal logic known as Timed Computation Tree Logic (TCTL) [2]. It extends the well-known simple branching-time temporal logic CTL – essentially a language to formalise nested reachability queries – with the ability to make assertions about the duration of time that passes along the runs of the system. The model checking problem for TCTL (over systems specified as timed automata) is PSPACE-complete [2], i.e. more difficult than the polynomial-time model checking for CTL (over finite

transition systems) [15]. Its expressive power in terms of structural properties is limited: similar to CTL, properties specifiable in TCTL are nested reachability queries expressed by a combination of a universal or existential quantification over an execution path and a simple temporal property of the form "something happens eventually / always / until something else happens".

For many verification tasks, such limited expressiveness far below regularity (i.e. definable in MSO) is sufficient. Yet in practice, for example when systems are composed of parallel and interacting components, additional expressiveness raised to full regularity may be needed [23]. In other situations, even full regularity is not enough as there is no formula of a temporal logic of regular expressiveness that e.g. describes the lack of underflows in FIFO or LIFO buffers of unbounded size [21]. One may argue that in practice, a buffer is always bounded but this makes the correctness property depend on the implementation. It should be clear that correctness properties should be formalisable independently of the system that is supposed to satisfy them, for otherwise formal verification could easily be achieved in general.

In order to extend the applicability of formal verification for real-time systems in situations where correctness is a structurally more complex property than what fits into TCTL, we have recently proposed Timed Recursive CTL (TRCTL) [12] which merges two extensions of CTL: the aforementioned one by real-time aspects that lifts CTL to TCTL is combined by an extension to properties that are specifiable using recursive property transformers (in the form of first-order functions[1]). This is taken from the untimed world where logics of high expressiveness have been studied for the same reasons as laid out here [18]. The high expressiveness comes at a price, both in terms of computational complexity as well as pragmatics. The syntax of logics like HFL [24] is based on the modal μ-calculus and a typed λ-calculus, and is therefore fairly inaccessible to non-experts and thus not usable at the forefront in system design and verification. Recursive CTL (RecCTL) has therefore been proposed to allow for a reasonable extension of expressive power beyond regularity whilst retaining as much intuitive syntax from CTL as possible.

The model checking problem for TRCTL over timed automata is 2EXPTIME-complete [12]. This may seem odd because that of TCTL is "only" PSPACE-complete, and the extension facilitated by recursive predicates (as least fixpoints of first-order functions) should raise the complexity intuitively by one exponential and result in EXPSPACE-completeness. However, the recursion operator lifts the restriction to a fixed system of nested reachability queries built into CTL's syntax. Hence, said recursion operator not only introduces predicates with unbounded recursion in the world of first-order functions, but also for ordinary predicates in temporal formulas. Thus, adding the recursion operator implicitly turns the base logic from TCTL into a timed variant of the modal μ-calculus. Such temporal logics which can express unbounded recursion – here in the form of least and greatest fixpoints of predicates – typically have model checking complexities that are complete for time classes.

The timed μ-calculus does not feature as prominently in the literature as its untimed counterpart, the modal μ-calculus [17], probably due to the combination of real-time operators and explicit fixpoint operators which may be unsellable to an ordinary user in formal verification. There are also syntactic variants in the literature that could be covered by the generic term *timed μ-calculus* [2, 16]. Going further into this is beyond the scope of this paper; we simply note that "the" timed μ-calculus (obtained from TRCTL as the restriction without first-order elements) has an EXPTIME-complete model checking problem [1] and can therefore be seen as a fragment of TRCTL of lower complexity, yet a regular one.

[1] See the formal definition of the syntax in Sect. 2.2 for an explanation of what "first-order" means here.

In this paper we investigate the question after the existence of a fragment of TRCTL whose expressiveness remains reasonably beyond regularity and whose model checking complexity is genuinely lower than that of full TRCTL (up to the current knowledge in complexity theory). We employ a syntactic restriction called tail-recursiveness which limits the ways that recursive properties can be defined. In untimed logics, tail-recursiveness leads to lower complexity [13], and it is characteristic of space rather than time complexity. The result in this paper therefore fits into what can be expected from previous work on timed and non-regular specification languages: the model checking problem for tail-recursive TRCTL is "only" EXPSPACE-complete, i.e. exactly one exponential worse than that of its untimed counterpart. The upper bound is established making use of the well-known region-graph abstraction [3]. The lower bound is established by a reduction from a suitable tiling problem.

The paper is organised as follows. In Sect. 2 we recall preliminaries on timed automata and TRCTL. In Sect. 3 we introduce tail-recursiveness, define the fragment under consideration here and argue why it can be model checked in exponential space. In Sect 4 we present the more elaborate lower bound construction. Sect. 5 contains some remarks on further work.

2 Preliminaries

2.1 Timed Automata

Timed Transition Systems. A *timed labelled transition system* (TLTS) over a finite set *Prop* of atomic propositions (and a single, anonymous[2] action) is a $\mathcal{T} = (\mathcal{S}, \to, s_0, \lambda)$ s.t.

- $\mathcal{S}$ is a set of *states* containing a designated starting state s_0,
- $\to \subseteq \mathcal{S} \times \mathcal{S} \cup \mathcal{S} \times \mathbb{R}^{\geq 0} \times \mathcal{S}$ is the transition relation, consisting of two kinds:
  - *discrete transitions* of the form $s \to t$ 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

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

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

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

$$s \overset{d}{\Longrightarrow} t \quad \text{iff} \quad \exists d_1, d_2 \in \mathbb{R}^{\geq 0}, s', t' \in \mathcal{S} \text{ s.t. } s \xrightarrow{d_1} s', s' \to t', t' \xrightarrow{d_2} t \text{ and } d = d_1 + d_2$$

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

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

Clock Constraints. Let $\mathcal{X} = \{\mathtt{x}, \mathtt{y}, \dots\}$ 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 conjunctive formulas of the form $\top$ or $\mathtt{x} \oplus c$ for $\mathtt{x} \in \mathcal{X}$, $c \in \mathbb{N}$ and $\oplus \in \{\leq, <, \geq, >, =\}$. We write $\mathtt{x} \in [c, c']$ for $\mathtt{x} \geq c \wedge \mathtt{x} \leq c'$, and similarly for open interval bounds.

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

[2] CTL-based logics are usually oblivious to action labels, whence we restrict ourselves to a single action.

- $\eta \models \top$ holds for any η,
- $\eta \models \varphi_1 \wedge \varphi_2$ if $\eta \models \varphi_1$ and $\eta \models \varphi_2$,
- $\eta \models \mathtt{x} \oplus c$ if $\eta(\mathtt{x}) \oplus c$ 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 $(\eta{+}d)(\mathtt{x}) = \eta(\mathtt{x}) + d$ for any $\mathtt{x} \in \mathcal{X}$, and $\eta|_R$ for the clock evaluation that is defined by $\eta|_R(\mathtt{x}) = 0$ for $\mathtt{x} \in R$ and $\eta|_R(\mathtt{x}) = \eta(\mathtt{x})$ otherwise.

Timed Automata. Again, since the CTL-based logics considered here are oblivious of action names, we introduce *timed automata* (TA) over a single anonymous action. Such a TA over *Prop* is an $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda)$ 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,
- $\iota : L \to CC(\mathcal{X})$ 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. 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,
- $\lambda : L \to 2^{Prop}$ labels each location with the set of atomic propositions that hold true in it.

The *index* $m(\mathcal{A})$ of $\mathcal{A}$ is the largest constant occurring in its invariants or guards. Its *size* is

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

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, resp. behaviour of a TA $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda)$ is given by a TLTS $\mathcal{T}_\mathcal{A}$ over the time domain $\mathbb{R}^{\geq 0}$ as follows.

- The state set is $\mathcal{S} = \{(\ell, \eta) \mid \ell \in L, \eta \in (\mathcal{X} \to \mathbb{R}^{\geq 0}) \text{ such that } \eta \models \iota(\ell)\}$, consisting of pairs of locations and clock evaluations that satisfy the location's invariant.
- The initial state is $s_0 = (\ell_0, \eta_0)$ where $\eta_0(\mathtt{x}) = 0$ for all $\mathtt{x} \in \mathcal{X}$.
- Delay transitions retain the 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 $\eta{+}d' \models \iota(\ell)$ for all $d' \leq d$.
- 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 $\eta \models g$ as well as $\eta|_R \models \iota(\ell')$.
- The propositional label of a state is that of its underlying location: $\lambda(\ell, \eta) = \lambda(\ell)$.
- Clock constraints hold in a state if they hold for its clocks: $(\ell, \eta) \models \chi$ iff $\eta \models \chi$.

In other words, a TA finitely represents a TLTS. However, not every TLTS is finitely representable. For a detailed introduction to timed automata we refer to the literature [3, 6]. Henceforth, we will only consider TLTS that arise from a TA. Consequently, we can always assume that the interpretation of clock constraints like $\mathtt{x} \leq 4$ in a TLTS is well-defined.

The Region Abstraction. There is a well-known abstraction of a TLTS $\mathcal{T}_\mathcal{A}$ into a finite LTS known as the *region graph* $\mathcal{R}_\mathcal{A}$ [3], used in decidability proofs for decision problems on TA.

In the following we only consider TLTS $\mathcal{T}_\mathcal{A}$ that arise from some TA $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda)$. The region abstraction is a mapping of such $\mathbb{R}^{\geq 0}$-TLTS into finite LTS. It is based on an equivalence relation $\simeq_m$, for $m \in \mathbb{N}$, on clock evaluations defined as follows.

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

Here, $\mathit{frac}(r)$ denotes the fractional part of a real number. It is easy to see that $\simeq_m$ is indeed an equivalence relation for any m. It is lifted to states of the TLTS $\mathcal{T}_\mathcal{A}$ in the most straight-forward way: $(\ell, \eta) \simeq_m (\ell', \eta')$ iff $\ell = \ell'$ and $\eta \simeq_m \eta'$.

We write $[\eta]_m$ for the equivalence class of η under $\simeq_m$ and likewise for $[(\ell, \eta)]_m$. When m is clear from the context we may also drop it and simply write $[\eta]$, resp. $[(\ell, \eta)]$.

Note that $\simeq_m$ is a bisimulation on the state space of $\mathcal{T}_\mathcal{A}$ w.r.t. the labelling and discrete and delay transitions: if $(\ell, \eta) \simeq_m (\ell', \eta')$ then we have $\lambda([(\ell, \eta)]) = \lambda([(\ell', \eta')])$ and for every ℓ'', η'': $[(\ell, \eta)] \to [(\ell'', \eta'')]$ iff $[(\ell', \eta')] \to [(\ell'', \eta'')]$. This is what makes it usable for an abstraction of the uncountable state space of $\mathcal{T}_\mathcal{A}$ into a finite discrete state space as follows.

The *region graph* $\mathcal{R}_\mathcal{A}$ of the TA $\mathcal{A}$ is the LTS $(\mathcal{S}, \to, s_0, \lambda)$ obtained as the quotient of $\mathcal{T}_\mathcal{A}$ under $\simeq_m$ with $m := m(\mathcal{A})$, together with an additional collapse of delay transitions for different delays into a single "*some-delay*" value τ. Its components are as follows.

- $\mathcal{S} = \{[(\ell, \eta)]_m \mid \ell \in L, \eta \in (\mathcal{X} \to \mathbb{R}^{\geq 0}), \eta \models \iota(\ell)\}$, and $s_0 = [(\ell_0, \eta_0)]_m$.
- Discrete transitions from one state to another are obtained by possibly delaying, then performing a discrete transition, then possibly delaying again afterwards. We have

$$[(\ell, \eta)]_m \to [(\ell', \eta')]_m \quad \text{if there are } d, d' \in \mathbb{R}^{\geq 0}, \hat{\eta}, \hat{\eta}' \text{ s.t. } (\ell, \eta) \xrightarrow{d_1} (\ell, \hat{\eta}) \to (\ell', \hat{\eta}') \xrightarrow{d_2} (\ell', \eta')$$

  for any $\ell, \ell' \in L$, $\eta, \eta' \in \mathcal{X} \to \mathbb{R}^{\geq 0}$.
- The propositional labelling is given as $\lambda([(\ell, \eta)]_m) = \lambda(\ell, \eta) = \lambda(\ell)$.

▶ **Proposition 1** ([3]). *Let $\mathcal{A}$ be a TA over n clocks with ℓ locations and of index m. Then $\mathcal{R}_\mathcal{A}$ is an (untimed) LTS of size $\ell \cdot 2^{\mathcal{O}(n(\log n + \log m))}$, i.e. exponential in $|\mathcal{A}|$, and there is a path $s_0 \xRightarrow{d_0} s_1 \xRightarrow{d_1} \dots$ in $\mathcal{T}_\mathcal{A}$ iff there is a path $[s_0] \to [s_1] \to \dots$ in $\mathcal{R}_\mathcal{A}$.*

2.2 Timed Recursive Computation-Tree Logic

TRCTL incorporates the two extensions from CTL to TCTL introducing real-time and to RecCTL introducing recursive predicates.

Syntax. Let *Prop* be a set of atomic propositions. Let $\mathcal{V}_1 = \{x, y, \dots\}$ be a set of propositional variables and $\mathcal{V}_2 = \{\mathcal{F}, \dots\}$ be a set of so-called *recursion* variables. Formulas of TRCTL are given by the following grammar.

$$\varphi ::= q \mid x \mid \chi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathtt{E}(\varphi\, \mathtt{U}_J\, \varphi) \mid \mathtt{A}(\varphi\, \mathtt{U}_J\, \varphi) \mid \Phi(\varphi, \dots, \varphi)$$
$$\Phi ::= \mathcal{F} \mid \mathtt{rec}\, \mathcal{F}(x_1, \dots, x_k).\, \varphi$$

where $q \in \mathit{Prop}$, J denotes an interval in $\mathbb{R}^{\geq 0}$ with rational bounds, χ is a clock constraint, and $x, x_i, y_i \in \mathcal{V}_1$, $\mathcal{F} \in \mathcal{V}_2$.

The (sub-)formulas derived from φ are called *propositional*, those derived from Φ are called *first-order*. We allow further Boolean operators like $\mathtt{tt}, \mathtt{ff}, \to$, etc. and temporal operators like $\mathtt{EF}, \mathtt{EG}, \mathtt{AF}, \mathtt{AG}$, etc. through their standard abbreviations. We also avoid parentheses through the standard precedence rules and remove empty tuples, i.e. we write $\mathtt{rec}\, \mathcal{F}.\varphi$ instead of $(\mathtt{rec}\, \mathcal{F}().\varphi)()$. We also consider $\mathtt{rec}\, \mathcal{F}.\varphi$ as a propositional rather than a first-order formula, because it results from the application of a first-order formula to zero arguments.

The grammar above allows non-well-formed formulas to be constructed, too. These need to be excluded using a stronger mechanism than context-free grammars. We refer to [12] for a formal definition of a (simple but tedious) typing system for well-formedness and introduce this notion intuitively instead: a well-formed formula obeys the following two restrictions.

- The numbers of formal parameters and arguments coincide, i.e. in a formula of the form $(\mathtt{rec}\,\mathcal{F}(x_1,\dots,x_n).\varphi)(\psi_1,\dots,\psi_m)$ we must have $n=m$. However, arguments can also be passed to a first-order formula of the form $\mathcal{F}$ where the parameters are not visible. We assume that each recursion variable is bound by the recursion operator at most once, whence, we can associate with each (bound) $\mathcal{F}$ an arity given by the number of parameters in its definition. This must then also match the number of arguments passed to it.
- The recursion operator is explained semantically via least fixpoints in function lattices. For this to be well-defined, each recursive call must occur positively in its defining body. Violations occur, e.g., in $\mathcal{F}().\neg\mathcal{F}$ or in $\mathtt{rec}\,\mathcal{F}.(\mathtt{rec}\,\mathcal{G}(x).\neg x)(\mathcal{F})$ where both recursion variables $\mathcal{F}$ and $\mathcal{G}$ appear to be used positively only, resp. not at all. Hence, a simple criterion like occurrence under an even number of negation symbols does not capture well-definedness as negative occurrences can be hidden in function applications.

Semantics. Formulas of TRCTL are interpreted over timed transition systems $\mathcal{T}=(\mathcal{S},\to,s_0,\lambda)$ (as arising from TA for example). A propositional formula φ denotes a set of states $[\![\varphi]\!]^{\mathcal{T}}\subseteq\mathcal{S}$, while a first-order formula with k formal parameters denotes a function $[\![\Phi]\!]^{\mathcal{T}}:(2^{\mathcal{S}})^k\to 2^{\mathcal{S}}$ that maps k sets of such states to a set of states. The semantics is given inductively, which is why environments α are needed in order to explain the meaning of free variables. Formally, α maps propositional variables x to sets of states and first-order variables $\mathcal{F}$ to functions as stated above. The semantics is then defined via

$$[\![q]\!]^{\mathcal{T}}_\alpha := \{s \mid q\in\lambda(s)\} \qquad [\![x]\!]^{\mathcal{T}}_\alpha := \alpha(x) \qquad [\![\chi]\!]^{\mathcal{T}}_\alpha = \{s \mid s\models\chi\}$$
$$[\![\neg\varphi]\!]^{\mathcal{T}}_\alpha := \mathcal{S}\setminus[\![\varphi]\!]^{\mathcal{T}}_\alpha \qquad [\![\varphi\vee\varphi]\!]^{\mathcal{T}}_\alpha := [\![\varphi]\!]^{\mathcal{T}}_\alpha\cup[\![\psi]\!]^{\mathcal{T}}_\alpha \qquad [\![\varphi\wedge\varphi]\!]^{\mathcal{T}}_\alpha := [\![\varphi]\!]^{\mathcal{T}}_\alpha\cap[\![\psi]\!]^{\mathcal{T}}_\alpha$$

and

$$\begin{aligned}
[\![\mathtt{E}(\varphi\,\mathtt{U}_J\,\varphi)]\!]^{\mathcal{T}}_\alpha &:= \{s \mid \text{ there is a path } \pi=s,\dots \text{ s.t. } \mathcal{T},\pi\models_\alpha \varphi\,\mathtt{U}_J\,\psi\}\\
[\![\mathtt{A}(\varphi\,\mathtt{U}_J\,\varphi)]\!]^{\mathcal{T}}_\alpha &:= \{s \mid \text{ for all paths } \pi=s,\dots \text{ we have } \mathcal{T},\pi\models_\alpha \varphi\,\mathtt{U}_J\,\psi\}\\
[\![\Phi(\varphi_1,\dots,\varphi_n)]\!]^{\mathcal{T}}_\alpha &:= [\![\Phi]\!]^{\mathcal{T}}_\alpha([\![\varphi_1]\!]^{\mathcal{T}}_\alpha,\dots,[\![\varphi_n]\!]^{\mathcal{T}}_\alpha)\\
[\![\mathcal{F}]\!]^{\mathcal{T}}_\alpha &:= \alpha(\mathcal{F})\\
[\![\mathtt{rec}\,\mathcal{F}(x_1,\dots,x_n).\,\varphi]\!]^{\mathcal{T}}_\alpha &:= \bigsqcap\{f:(2^{\mathcal{S}})^n\to 2^{\mathcal{S}} \mid \text{ for all } T_1,\dots,T_n\subseteq\mathcal{S} \text{ we have}\\
&\qquad\qquad [\![\varphi]\!]^{\mathcal{T}}_{\alpha[\mathcal{F}\mapsto f,x_1\mapsto T_1,\dots x_n\mapsto T_n]}\subseteq f(T_1,\dots,T_n)\}
\end{aligned}$$

where $(\bigsqcap_{i\in I} f_i)(T_1,\dots,T_n) := \bigcap_{i\in I} f_i(T_1,\dots,T_n)$ and the satisfaction of a $\mathtt{U}$-property by a non-Zeno path $\pi=s_0\overset{d_0}{\Longrightarrow}s_1\overset{d_1}{\Longrightarrow}s_2\overset{d_2}{\Longrightarrow}\dots$ in the TLTS is given as follows. We have $\pi\models_\alpha\varphi\,\mathtt{U}_J\,\psi$ iff

$$\begin{aligned}
&\exists i\geq 0,\exists d\in[0,d_i],\exists s' \text{ s.t. } s_i\overset{d}{\Longrightarrow}s' \text{ and } (\sum_{h=0}^{i} d_i)+d\in J \text{ and } s'\in[\![\psi]\!]^{\mathcal{T}}_\alpha \text{ and}\\
&\qquad\forall j<i,\forall d'\in[0,d_j],\forall s' \text{ s.t. } s_j\overset{d'}{\Longrightarrow}s' \text{ we have } s'\in[\![\varphi\vee\psi]\!]^{\mathcal{T}}_\alpha \text{ and}\\
&\qquad\forall d'\in[0,d),\forall s' \text{ s.t. } s_i\overset{d'}{\Longrightarrow}s' \text{ we have } s'\in[\![\varphi\vee\psi]\!]^{\mathcal{T}}_\alpha.
\end{aligned}$$

This may seem odd at first glance but it is in fact standard to interpret an Until operator in this way in the real-time setting, cf. [16, 6]. The sum on the right-hand side is simply used to express the reaching of some state in the future by delay steps along multiple transitions, hence the time that passes up to this step is being added up. The other possibly unintuitive feature is the seemingly weak assertion on s' (under the universal quantification) to satisfy φ *or* ψ, where one may assume it to have to satisfy φ. First note that allowing such "earlier" moments to also satisfy ψ instead of φ is not harmful to the intuitive meaning of $\varphi \,\mathtt{U}\, \psi$: if ψ holds at some point but also earlier as well, then it still holds at some point. In a discrete-time setting there is always a first moment at which ψ holds, and it suffices when all moments before that satisfy φ. However, in the real-time setting there may not be a first moment for ψ to hold. So it is in fact necessary to allow these earlier moments to also satisfy the Until's right argument. This ensures, for example, that a formula like $\mathtt{E}(\mathtt{x}{=}0 \,\mathtt{U}\, \mathtt{x}{>}0)$ is satisfiable. Note that, after a moment satisfying $\mathtt{x} = 0$, there is no first moment satisfying $\mathtt{x} > 0$, but intuitively the formula should be satisfiable.

For a closed formula φ and arbitrary $s \in \mathcal{S}$ 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 $\mathcal{T}, s_0 \models \varphi$.

Examples. TRCTL is able to express structurally complex properties of real-time systems. We give two examples which show how the recursion operator can be used to create combinations of temporal formulas that could not be expressed in logics of regular expressiveness only.

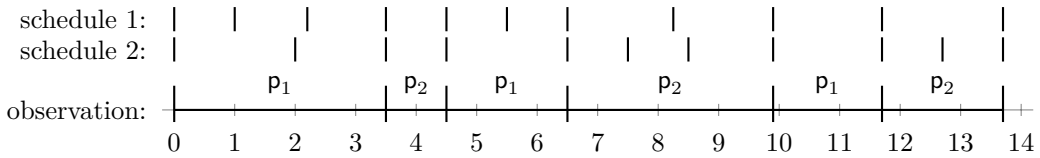
▶ **Example 2.** Consider a TLTS over propositions including $\{\mathsf{r}, \mathsf{g}\}$ which signal the request of a resource respectively the granting of such a request. The TRCTL formula

$$\big(\,\mathtt{rec}\,\mathcal{F}(x,y).(x \to y) \wedge \mathcal{F}(\mathtt{EF}^{\leq 2}x, \mathtt{EF}^{\leq 3}y)\big)(\mathsf{r},\mathsf{g})$$

then states "whenever a request is issued after at most $2n$ time units, then a grant is issued after at most $3n$ time units (for the same n)". To see that this is indeed expressed by the formula one only needs three principles: (I) unfolding of recursive definitions and (II) replacement of parameter variables by arguments, and (III) the temporal simplification rule $\mathtt{EF}^{\leq c}\mathtt{EF}^{\leq d}\psi \equiv \mathtt{EF}^{\leq c+d}\psi$. To keep the calculation short we identify $\mathcal{F}$ with $\mathtt{rec}\,\mathcal{F}(x,y).(x \to y) \wedge \mathcal{F}(\mathtt{EF}^{\leq 2}x, \mathtt{EF}^{\leq 3}y)$. Then we have

$$\begin{aligned}
\mathcal{F}(\mathsf{r},\mathsf{g}) &\equiv (\mathsf{r} \to \mathsf{g}) \wedge \mathcal{F}(\mathtt{EF}^{\leq 2}\mathsf{r}, \mathtt{EF}^{\leq 3}\mathsf{g}) \\
&\equiv (\mathsf{r} \to \mathsf{g}) \wedge (\mathtt{EF}^{\leq 2}\mathsf{r} \to \mathtt{EF}^{\leq 3}\mathsf{g}) \wedge \mathcal{F}(\mathtt{EF}^{\leq 2}\mathtt{EF}^{\leq 2}\mathsf{r}, \mathtt{EF}^{\leq 3}\mathtt{EF}^{\leq 3}\mathsf{g}) \\
&\equiv (\mathsf{r} \to \mathsf{g}) \wedge (\mathtt{EF}^{\leq 2}\mathsf{r} \to \mathtt{EF}^{\leq 3}\mathsf{g}) \wedge \mathcal{F}(\mathtt{EF}^{\leq 4}\mathsf{r}, \mathtt{EF}^{\leq 6}\mathsf{g}) \\
&\equiv (\mathsf{r} \to \mathsf{g}) \wedge (\mathtt{EF}^{\leq 2}\mathsf{r} \to \mathtt{EF}^{\leq 3}\mathsf{g}) \wedge (\mathtt{EF}^{\leq 4}\mathsf{r} \to \mathtt{EF}^{\leq 6}\mathsf{g}) \wedge \mathcal{F}(\mathtt{EF}^{\leq 6}\mathsf{r}, \mathtt{EF}^{\leq 9}\mathsf{g}) \\
&\equiv \quad \cdots \quad \equiv \bigwedge_{n \geq 0} \mathtt{EF}^{\leq 2n}\mathsf{r} \to \mathtt{EF}^{\leq 3n}\mathsf{g}.
\end{aligned}$$

▶ **Example 3.** Take a timed system in which a scheduler governs the execution of two different processes. We assume that proposition p_i, $i \in \{1,2\}$ holds whenever process i is active, that at any moment exactly one of them holds, and that the execution of a process takes between 1 and 2 time units. A possible trace of such a system w.r.t. only the two propositions p_1 and p_2 in real time is represented by the bottom line in this picture:



The trace divides the real line into intervals during which either of the two propositions in question holds. Clearly, from this trace we can derive that the scheduler finished an execution of process 1 and started an execution of process 2 at time point 3.5 for example. Only the switches from one process to another are visible. The finishing of process i and subsequent rescheduling of it cannot be inferred from these propositions alone. All that is known, for example, is that between time moments 0 and 3.5, there must have been at least 2 and at most 3 executions of process 1. Thus, such a trace can result from several different schedulings; two of these are depicted here on top. In schedule 1, three instances of process 1 have been completed before time point 3.5, in schedule 2 only two of them have been run, etc.

Now suppose that there is an additional constraint stating that at any moment, process 2 may never have been scheduled more often than process 1. Note that schedule 1 satisfies this property but schedule 2 does not since, at time point 9.9, only three instances of process 1 have been completed but already four instances of process 2 are done.

The TRCTL formula

$$\neg\Big(\big(\,\mathtt{rec}\,\mathcal{F}(x).\mathtt{E}(\mathsf{p}_2\,\mathtt{U}^{\geq 1}\,x) \vee \mathtt{E}(\mathsf{p}_1\,\mathtt{U}^{\leq 2}\,\mathcal{F}(\mathcal{F}(x)))\big)(\mathtt{tt})\Big)$$

guarantees the absence of such faulty schedulings. It states that it is not possible to find a path and its division into intervals of length at least 1, resp. at most 2, depending on whether p_1 or p_2 holds at the moment, such that at some point the number of p_2-intervals has exceeded the number of p_1-intervals seen so far.

To understand how this is expressed it is probably best to remember that the context-free grammar $F \to b \mid aFF$ generates all (minimal) words w s.t. $|w|_b > |w|_a$ but $|v|_b \leq |v|_a$ for any prefix v of w. Note how the formula above follows exactly this structure. By unfolding the recursion and replacing arguments successively, $\mathcal{F}(\mathtt{tt})$ can be seen to be equivalent to a disjunction of nested EU-formulas like

$$\mathtt{E}(\mathsf{p}_1\,\mathtt{U}^{\leq 2}\,\mathtt{E}(\mathsf{p}_1\,\mathtt{U}^{\leq 2}\,\mathtt{E}(\mathsf{p}_2\,\mathtt{U}^{\geq 1}\,\mathtt{E}(\mathsf{p}_1\,\mathtt{U}^{\leq 2}\,\mathtt{E}(\mathsf{p}_2\,\mathtt{U}^{\geq 1}\,\mathtt{E}(\mathsf{p}_2\,\mathtt{U}^{\geq 1}\,\mathtt{E}(\mathsf{p}_2\,\mathtt{U}^{\geq 1}\,\mathtt{tt})))))))$$

which is satisfied by the trace presented above as schedule 2 shows. Each such disjunct demands one more occurrence of p_2- than p_1-intervals.

3 The Tail-Recursive Fragment of Timed Recursive CTL

3.1 The Syntactical Restriction

In functional programming, a definition of a recursive function is tail recursive if the return value of a function is the return value of a recursive call without alterations. Tail-recursive functions are often more efficient to evaluate since one does not need to remember intermediate variable bindings. This concept also yields improved efficiency in the evaluation of formulas defined via fixpoints, cf. e.g. [13, 9]. For this, it is crucial that the interplay between the fixpoints and the recursive definitions is not too complex.

$$\frac{}{\emptyset \vdash_{\mathsf{tr}} p} \qquad \frac{}{\emptyset \vdash_{\mathsf{tr}} x} \qquad \frac{}{\emptyset \vdash_{\mathsf{tr}} \chi} \qquad \frac{}{\{\mathcal{F}\} \vdash_{\mathsf{tr}} \mathcal{F}} \qquad \frac{\emptyset \vdash_{\mathsf{tr}} \varphi}{\emptyset \vdash_{\mathsf{tr}} \neg\varphi} \qquad \frac{\mathcal{V} \vdash_{\mathsf{tr}} \varphi_1 \qquad \mathcal{V}' \vdash_{\mathsf{tr}} \varphi_2}{\mathcal{V} \cup \mathcal{V}' \vdash_{\mathsf{tr}} \varphi_1 \vee \varphi_2}$$

$$\frac{\emptyset \vdash_{\mathsf{tr}} \varphi_1 \qquad \mathcal{V} \vdash_{\mathsf{tr}} \varphi_2}{\mathcal{V} \vdash_{\mathsf{tr}} \varphi_1 \wedge \varphi_2} \qquad \frac{\emptyset \vdash_{\mathsf{tr}} \varphi_1 \qquad \mathcal{V} \vdash_{\mathsf{tr}} \varphi_2}{\mathcal{V} \vdash_{\mathsf{tr}} \mathtt{E}(\varphi_1 \,\mathtt{U}_J\, \varphi_2)} \qquad \frac{\emptyset \vdash_{\mathsf{tr}} \varphi_1 \qquad \emptyset \vdash_{\mathsf{tr}} \varphi_2}{\emptyset \vdash_{\mathsf{tr}} \mathtt{A}(\varphi_1 \,\mathtt{U}_J\, \varphi_2)}$$

$$\frac{\mathcal{V} \vdash_{\mathsf{tr}} \Phi \qquad \emptyset \vdash_{\mathsf{tr}} \varphi_1 \qquad \cdots \qquad \emptyset \vdash_{\mathsf{tr}} \varphi_n}{\mathcal{V} \vdash_{\mathsf{tr}} \Phi(\varphi_1, \ldots, \varphi_n)} \qquad \frac{\mathcal{V} \vdash_{\mathsf{tr}} \varphi}{\mathcal{V} \setminus \{\mathcal{F}\} \vdash_{\mathsf{tr}} \mathtt{rec}\, \mathcal{F}(x_1, \ldots, x_m).\varphi}$$

Figure 1 Derivation rules for establishing tail-recursiveness. The sets $\mathcal{V}$ and $\mathcal{V}'$ denote the set of free fixpoint variables of the formula in question.

In our setting this means that recursion variables cannot appear in an operand setting, which is the equivalent to the above stipulation that the return value of a function is the return value of the recursive call, emphwithout alterations. Moreover, we also can have (almost) no branching introduced by boolean alternation, so at most one subformula of a formula of the form $\varphi_1 \wedge \varphi_2$ can have free recursion variables. If this is satisfied, the formula without free variables can be evaluated first in a suitable model-checking procedure (cf. [13]) until the recursion resumes in the other subformula. Note that U-formulas introduce hidden branching through their definition. The reason for this stipulation is that nondeterminism and universal nondeterminism introduce branching in the flow of a program if both branches contain recursive calls. However, it is well-known that space complexity classes from PSPACE and upwards admit free nondeterminism via Savitch's Theorem [20], hence one kind of nondeterminism can be mixed with recursive calls. Boolean alternation, however, can not be mixed with recursion without breaking this property. Since we make use of Savitch's Theorem to relax the requirements on disjunctions, we restrict boolean alternation. Also, this means that we have to restrict the use of negation, which would turn nondeterministic branching into universal branching. The above considerations are condensed into the derivation system in Fig. 1, giving the following definition:

▶ **Definition 4.** *A* TRCTL *formula φ is called* tail recursive *if the statement $\mathcal{V} \vdash_{\mathsf{tr}} \varphi$ for some, potentially empty, set of recursion variables $\mathcal{V}$ can be derived in the derivation system given in Fig. 1. We write* trTRCTL *for the fragment of all tail-recursive formulas.*

The derivation system in Fig. 1 is to be understood as follows: The set $\mathcal{V}$ in front of the $\vdash$ simply collects the set of free recursion variables of the subformula in question. For example, the subformula p has no free recursion variables, since it is a proposition, and neither does the subformula x, since x is not a recursion variable. The system also enforces the above stipulations: a formula directly under a negation can have no free recursion variables, but note that something like $\neg\mathcal{F}.\ p \vee \mathtt{E}(q \,\mathtt{U}_J\, \mathcal{F})$ is permitted. Disjunctions can contain free recursion variables on both sides, while conjunctions, including those introduced by U formulas, may contain free variables only on one side of the conjunction. The reason for this is that the subformula that is closed w.r.t. recursion can be evaluated first in a non-recursive fashion, and then recursion can proceed in a tail-recursive fashion on the other side. Hence, $\mathtt{rec}\,\mathcal{F}.\mathtt{E}(\mathtt{z} < 3 \,\mathtt{U}\, \mathcal{F})$ is tail-recursive, but $\mathtt{rec}\,\mathcal{F}(x).\mathcal{F}(p) \wedge \mathcal{F}(x \vee \mathtt{z} \leq 2)$ is not since it contains the recursion variable $\mathcal{F}$ on both sides of the conjunction.

Finally, applications may not contain subformulas with free recursion variables on the operand side. Hence, $\mathtt{rec}\,\mathcal{F}(x).x \wedge \mathcal{F}(\mathcal{F}(\mathtt{E}(\mathtt{z} \leq 3)\,\mathtt{U}\,x))$ is not tail recursive, and neither is the formula constructed in Ex. 3. However, the one from Ex. 2 is tail recursive.

3.2 Model Checking in Exponential Space

Tail recursiveness can be applied to the untimed logic RecCTL resulting in the fragment trRecCTL. Using the untiming construction we can then reduce the model checking problem for trTRCTL over TA to that of trRecCTL (over an exponentially larger LTS) whose complexity is not difficult to estimate.

▶ **Theorem 5.** *Model checking* trRecCTL *is in PSPACE.*

Proof. The model checking problem for (untimed, non-tail-recursive) RecCTL is known to be EXPTIME-complete [11]. The argument for the upper bound uses a conceptually simple polynomial translation into HFL^1, the first-order fragment of Higher-Order Fixpoint Logic whose model checking problem is known to be EXPTIME-complete [5]. The translation from RecCTL to HFL^1, when applied to a tail-recursive formula, also produces a formula of tail-recursive HFL^1. The model checking problem for this is known to be easier, namely only PSPACE-complete [13], which establishes the claim. ◀

We now lift this to an upper bound for trTRCTL via standard constructions.

▶ **Theorem 6.** *The* trTRCTL *model checking problem over TA is decidable in* EXPSPACE.

Proof. Let $\varphi \in$ trTRCTL and $\mathcal{A}$ be a TA not using the clock $\mathtt{z}$. We construct an LTS $\mathcal{R}^{\varphi}_{\mathcal{A}^{\mathtt{z}}}$ by extending the original region graph $\mathcal{R}_{\mathcal{A}}$ for $\mathcal{A}$ to make clock values visible to the formula.

- For each state $[(\ell,\eta)]$ and each $c \leq m(\varphi)$, add the proposition $p_{\mathtt{z}\oplus c}$ to $\lambda([(\ell,\eta)])$ if $\eta \models \mathtt{z} \oplus c$ for $\oplus \in \{\leq, <, \geq, >, =\}$.
- For each state $[(\ell,\eta)]$ introduce a new state $s_{[(\ell,\eta)]}$ with the sole label $\{r_{\mathtt{z}}\}$, and add transitions $[(\ell,\eta)] \to s_{[(\ell,\eta)]} \to [(\ell, \eta|_{\{\mathtt{z}\}})]$.

The formula $\varphi^{\mathtt{z}}$ results from φ by replacing each subformula of the form

- χ by p_χ,
- $\mathtt{E}(\psi_1\,\mathtt{U}_{[c,d]}\,\psi_2)$ by $\mathtt{EX}(r_{\mathtt{z}} \wedge \mathtt{EXE}((\neg r_{\mathtt{z}} \wedge \psi_1)\,\mathtt{U}\,(\neg r_{\mathtt{z}} \wedge p_{\mathtt{z}\in[c,d]} \wedge \psi_2)))$,
- $\mathtt{A}(\psi_1\,\mathtt{U}_{[c,d]}\,\psi_2)$ by $\mathtt{EX}(r_{\mathtt{z}} \wedge \mathtt{EXA}((\neg r_{\mathtt{z}} \to \psi_1)\,\mathtt{U}\,(\neg r_{\mathtt{z}} \to p_{\mathtt{z}\in[c,d]} \wedge \psi_2)))$.

For open intervals on one side, the p-propositions are amended accordingly to $p_{\mathtt{z}>c}$ etc.

We observe that $\varphi^{\mathtt{z}}$ is a formula of (untimed) tail-recursive RecCTL that is constructible in time $\mathcal{O}(|\varphi|)$, and $\mathcal{R}^{\varphi}_{\mathcal{A}^{\mathtt{z}}}$ is an (untimed) LTS of size at most (singly) exponential in $|\mathcal{A}|$ and $m(\varphi)$ and also constructible in such time. It is then standard to show, by induction on the structure of φ, that $\mathcal{T}_{\mathcal{A}} \models \varphi$ iff $\mathcal{R}^{\varphi}_{\mathcal{A}^{\mathtt{z}}} \models \varphi^{\mathtt{z}}$. This establishes an exponential reduction from trTRCTL model checking to trRecCTL model checking and, thus, an EXPSPACE bound on the former due to Thm. 5. ◀

4 An Exponential Space Lower Bound for Model Checking

The aim of this section is to provide a lower bound on model checking trTRCTL, matching the exponential-space upper bound in Thm. 6. For this, we first introduce the *exponential corridor tiling problem* ExpTiling, known to be EXPSPACE-complete.

4.1 Exponential Space Complexity

A *tiling system* is a $\mathcal{W} = (T, H, V, t_I, t_F)$ s.t. T is a finite set of tile types, $H, V \subseteq T \times T$ are two binary relations on T called *horizontal*, resp. *vertical matching relation*, and t_0 and t_{fin} are two designated so-called *initial* and *final* tiles.

A $C \subseteq \mathbb{N} \times \mathbb{N}$ is called *closed*, if for all $(i, j) \in C$ we have:

- if $i > 0$ then $(i-1, j) \in C$, and
- if $j > 0$ then $(i, j-1) \in C$.

A (valid) $\mathcal{W}$-*tiling* of such a closed subspace (for the tiling system $\mathcal{W}$ above) is a $\tau : C \to T$ that satisfies the following properties.

- $\tau(0,0) = t_I$,
- for all $(i+1, j) \in C$ we have $(\tau(i,j), \tau(i+1,j)) \in H$,
- for all $(i, j+1) \in C$ we have $(\tau(i,j), \tau(i,j+1)) \in V$, and
- there are i, j s.t. $\tau(i,j) = t_F$.

The *exponential corridor tiling problem* (ExpTiling) is the following.

given: a tiling system system $\mathcal{W} = (T, H, V, t_I, t_F)$ and a number n encoded unarily
decide: is there an m and a valid $\mathcal{W}$-tiling τ of the space $[2^n] \times [m]$?

In this case, we simply also say that there is a valid $\mathcal{W}$-tiling on the 2^n-corridor.

Note that $|T|^{2^n}$ is an upper bound on the minimal m witnessing the existence of a $\mathcal{W}$-tiling. Also, we can always assume that t_{fin} is placed in the final row $m-1$, as any closed subspace of a correctly tiled space which includes t_{fin} can also be given a valid $\mathcal{W}$-tiling.

Intuitively, the problem ExpTiling asks for the existence of a run of a nondeterministic, exponential-space bounded Turing Machine such that the configurations are abstractly represented as rows of tiles of width 2^n. The vertical matching relation in the tiling assures that each following configuration, resp. row, matches the one below according to a finite set of rules (which can be used to model the local rewriting behaviour of a Turing Machine). The horizontal matching relation is needed in order to assure that local transformations in a Turing Machine configuration only happen in a single place, namely where the tape head is located. A more detailed exposition and explanation of the connection between Turing Machine runs and tilings can be found in [14, Sect. 11.1].

▶ **Proposition 7** ([22]). *The problem ExpTiling is* EXPSPACE*-complete.*

We remark that ExpTiling is also EXPSPACE-hard when n is given in binary coding but the upper bound would not hold anymore. Moreover, the reduction to model checking trTRCTL presented below relies on the ability to write down clock constraints like $\mathtt{x} \leq 2^n - 1$ in polynomial time using binary encoding. This would be rather difficult for n encoded binarily.

4.2 The Reduction

Given a tiling system $\mathcal{W} = (T, H, V, t_I, t_F)$ with designated initial and final tiles $t_I, t_F \in T$, and a number $n \in \mathbb{N}$ encoded unarily, we construct – in time polynomial in $|\mathcal{W}|$ and n – a timed automaton $\mathcal{A}_{\mathcal{W}}$ with some location t_0, and a trTRCTL formula $\varphi_{\mathcal{W},n}$ s.t.

$$\mathcal{T}_{\mathcal{W}}, [(t_0, \mathtt{x} \mapsto 0)] \models \varphi_{\mathcal{W},n} \qquad \text{iff} \qquad \text{there is a valid } \mathcal{W}\text{-tiling on the } 2^n\text{-corridor}$$

where $\mathcal{T}_{\mathcal{W}}$ is the TLTS associated with the timed automaton $\mathcal{A}_{\mathcal{W}}$. The single clock $\mathtt{x}$ involved in this construction is never used in the TA's transitions or locations. Instead it only occurs in $\varphi_{\mathcal{W},n}$ in order to state that something happens along runs during the first $2^n - 1$ units.

In order to keep $\varphi_{\mathcal{W},n}$ tail-recursive, we cannot mimic the reduction from a similar problem showing 2-EXPTIME-hardness of model checking the full logic TRCTL. In that reduction, the constructed formula employs a recursion subformula $\mathtt{rec}\,\mathcal{F}(s,t).\ldots$ with two parameters s and t that encode, respectively, the indices of the coordinates of a tile. But then we would have to state that the tile located at (s,t) matches its right neighbour horizontally *and* its neighbour above vertically which leads to a body of the recursion formula, roughly containing something like $\mathcal{F}(s',t) \wedge \mathcal{F}(s,t')$ which renders the entire formula non-tail-recursive.

Instead, the trick is to construct $\varphi_{\mathcal{W},n}$ such that it uses unary recursion formulas of the form $\mathtt{rec}\,\mathcal{F}(r)$ with a single propositional variable r only, interpreted as a set of states of $\mathcal{T}_{\mathcal{W}}$, encoding an entire row of a possible $\mathcal{W}$-tiling. For this we simply let $\mathcal{A}_{\mathcal{W}}$ consist of $|T|$ many locations, arranged in a full clique such that, at any moment in time, a transition from any t to any t' (including t itself) is possible. We assume that $T = \{t_0, \ldots, t_{k-1}\}$ for some $k \in \mathbb{N}$, as we will need a total order on T later on. We also use T as atomic propositions and let each location t satisfy the proposition t uniquely. Invariants or guards are not needed. Hence, a run through $\mathcal{T}_{\mathcal{W}}$ can freely traverse through the locations in T and change between them at any moment in time.

The crucial intuition for this reduction is the following: a valid $\mathcal{W}$-tiling of the $[2^n] \times [m]$-corridor exists for some $m \geq 1$, iff there is a sequence $R_0, \ldots, R_{m-1}$ of *rows*, i.e. $\mathcal{W}$-tilings of the $[2^n] \times [1]$-corridor each, such that vertical matching is guaranteed between them. I.e. if $R_i = t_{i,0}, \ldots, t_{i,2^n-1}$ and $R_{i+1} = t_{i+1,0}, \ldots, t_{i+1,2^n-1}$ then $(t_{i,j}, t_{i+1,j}) \in V$ for all $j \in [2^n]$. Within each row, the horizontal matching relation needs to be obeyed of course.

The existence of such a sequence can be expressed by a recursion formula that, intuitively, takes an initial row and, for as long as the current row is not final, generates vertically matching successor rows and continues the search with one of them. For this we need to encode such rows as formulas. Propositional formulas are interpreted as sets of states in $\mathcal{T}_{\mathcal{W}}$, i.e. objects of the form $[(t, \mathtt{x} \mapsto v)]$, since the locations are just the tiles from T and the only clock that is used here is $\mathtt{x}$. This gives rise to a canonical representation of such a row r_i as the set containing exactly the pairs (t_{i_j}, j) for $j = 0, \ldots, 2^n - 1$. For simplicity we write (t, x) instead of $[(t, \mathtt{x} \mapsto x)]$. In the following, x will implicitly be understood as a value of clock $\mathtt{x}$. Moreover, we will write $[\![\cdot]\!]^\alpha$ for $[\![\cdot]\!]^\alpha_{\mathcal{T}_{\mathcal{W}}}$.

▶ **Definition 8.** *A propositional formula ψ is said to be a (representation of) a* row candidate *(under α) if there are $t_{i_0}, \ldots, t_{i_{2^n-1}} \in T$ s.t. $[\![\psi]\!]_\alpha = \{(t_{i_0}, 0), (t_{i_1}, 1), \ldots, (t_{i_{2^n-1}}, 2^n - 1)\}$.*

A row *is such a row candidate that additionally satisfies: $(t_{i_j}, t_{i_{j+1}}) \in H$ for all $j \in [2^n - 1]$.*

Let *first* $:= t_0 \wedge \mathtt{EF}^*_{=1}(\mathtt{x} = 2^n - 1)$ where $\mathtt{EF}^*_{=1}\psi := \mathtt{rec}\,\mathcal{G}.\psi \vee \mathtt{EF}_{=1}\mathcal{G}$ expresses that some state satisfying ψ can be reached in an integer interval of time. It is satisfied by a state (t, x) iff $t = t_0$ and $x \in [2^n]$. The second conjunct ensures that x must be an integer value. Hence, any state in the set defined by *first* must combine these two properties. Since exactly the states $(t_0, 0), \ldots, (t_0, 2^n - - 1)$, satisfy both properties, *first* defines the set $\{(t_0, 0), \ldots, (t_0, 2^n - 1)\}$ or, likewise, it represents the row candidate $t_0, \ldots, t_0$.

A row is a row candidate in which adjacent tiles match w.r.t. H. This is also easy to express:

$$\mathit{row}(r) := \mathtt{AG}_{\leq 2^n - 1}\big(r \to \bigwedge_{(t,t') \notin H} t \to \mathtt{AG}_{=1}(r \to \neg t')\big)$$

Here we use that $\mathcal{A}_{\mathcal{W}}$ forms a clique and uses no clocks. Hence, any state (t', x') of $\mathcal{T}_{\mathcal{W}}$ is reachable from any state (t, x) for as long as $x \leq x'$.

▶ **Lemma 9.** *Let α be an interpretation mapping the propositional variable r to a row candidate $\alpha(r)$. Then $(t_0, 0) \in [\![row(r)]\!]_\alpha$ iff $\alpha(r)$ is in fact a row.*

There is no reason other than notational canonicity to choose t_0 as the location in which $row(r)$ is being evaluated. The statement also holds for any other location. What is important for the following arguments is that it is evaluated in a state with clock value 0. In a state (t, x) with $x > 0$ one simply cannot access all the tiles contained in a row candidate because time only moves forward and states (t', x') with $x' < x$ are not reachable from (t, x).

The next construction is more involved. Ultimately, we want to enumerate all possible row candidates in order to choose a next row in the iterative process described above. There is a standard way of getting from one row candidate to a canonical next one. Starting with the row candidate represented by *first*, we obtain the next one by incrementing a number represented in base-$|T|$ coding, making use of the total order on T given by the indices which makes a row candidate $t_{i_0}, \dots, t_{i_{2^n-1}}$ as a base-$|T|$ number with 2^n many digits. However, here we assume that the least significant digit is on the right, i.e. it is the one indexed $2^n - 1$. The reason for this is that the value of a digit in an incremented number depends on the values of the digits of lesser significance. In trTRCTL and a TA with no clock resets we can only access the future, yet not the past. Letting earlier time moments represent digits of higher significance allows for a simpler encoding of a base-$|T|$ increment operation. Given any row candidate $R_i = t_{i,0}, \dots, t_{i,2^n-1}$, the next one R_{i+1} is obtained using the well-known mechanism of incrementing a number represented in base $|T|$:

$$t_{i+1,j} = \begin{cases} t_0 & , \text{ if } t_{i,h} = t_{k-1} \text{ for all } h = j, \dots, 2^n - 1, \\ t_{m+1} & , \text{ if } t_{i,j} = t_m, m < k-1 \text{ and } t_{i,h} = t_{k-1} \text{ for all } h = j+1, \dots, 2^n - 1, \\ t_{i,j} & , \text{ if there is } h > j \text{ s.t. } t_{i,h} \neq t_{k-1}. \end{cases}$$

This can straightforwardly be formalised as follows.

$$\begin{aligned} next(r) := \big(t_0 \wedge \mathtt{AG}^*_{=1}(r \to t_{k-1})\big) \vee \Big(\bigvee_{m=0}^{k-2} t_{m+1} \wedge \mathtt{EF}_{=0}(r \wedge t_m)\Big) \\ \vee \Big(\bigvee_{m=0}^{k-1} t_m \wedge \mathtt{EF}_{=0}(r \wedge t_m)\Big) \wedge \mathtt{EF}^+_{=1}(r \wedge \neg t_{k-1}) \end{aligned}$$

where $\mathtt{EF}^+_{=1}\psi := \mathtt{EF}_{=1}\mathtt{EF}^*_{=1}\psi$ and $\mathtt{AG}^*_{=1}\psi := \neg\mathtt{EF}^*_{=1}\neg\psi$.

▶ **Lemma 10.** *Define a sequence of sets of states in $\mathcal{T}_\mathcal{W}$ as follows: $R_0 := [\![first]\!]$, $R_{i+1} := [\![next(r)]\!]_{[r \mapsto R_i]}$.*

a) *R_i is a row candidate for all $i \geq 0$.*

b) *Let $m := |T|^{2^n}$. The sets $R_0, \dots, R_{m-1}$ are pairwise different. Consequently, the sequence $R_0, R_1, \dots$ constitutes an enumeration of all possible row candidates for the given $\mathcal{W}$.*

Using this we can facilitate a search for a row (satisfying some formula $\psi(r)$) by enumerating all row candidates in a recursive iteration and terminating it when a proper row r satisfying $\psi(r)$ has been found.

$$\exists^{\mathsf{row}} r.\psi(r) := \Big(\mathtt{rec}\, \mathcal{G}(r).row(r) \wedge (\psi(r) \vee \mathcal{G}(next(r)))\Big)(first)$$

▶ **Lemma 11.** *Let $\psi(r)$ be a formula. We have $(t_0, 0) \in [\![\exists^{\mathsf{row}} r.\psi(r)]\!]$ iff there is a (representation of a) row R such that $(t_0, 0) \in [\![\psi(r)]\!]_{[r \mapsto R]}$, i.e. that satisfies ψ. Moreover, $\exists^{\mathsf{row}} r.\psi(r)$ is tail-recursive if $\psi(r)$ is so.*

A valid $\mathcal{W}$-tiling is comprised of a sequence of rows, starting with an initial one and ending in a final one. The initial one is such that its first tile, i.e. in position 0, is t_I; a final row is one that contains the final tile t_F. Both are easily specified as follows.

$$\mathit{init}(r) := \mathtt{AG}_{=0}(r \to t_I) \qquad \mathit{final}(r) := \mathtt{EF}^*_{=1}(r \wedge t_F)$$

▶ **Lemma 12.** *Let R be a (representation of a) row.*

a) $(t_0, 0) \in [\![\mathit{init}(r)]\!]_{[r \mapsto R]}$ *iff* $(t_I, 0) \in R$*, i.e. R starts with the initial tile.*

b) $(t_0, 0) \in [\![\mathit{final}(r)]\!]_{[r \mapsto R]}$ *iff there is* $i \in [2^n - 1]$ *s.t.* $(t_F, i) \in R$*, i.e. R contains the final tile.*

We now construct the overall formula $\varphi_{\mathcal{W},n}$ such that, procedurally thinking, it facilitates a search through the space of rows in order to decide the existence of a valid $\mathcal{W}$-tiling. It starts with some initial row, generates successors (which need to match vertically in all positions of these rows), until a final row has been found. All that is needed at this point is a formula that takes two rows r, r' and decides whether r' can be placed above r in a valid $\mathcal{W}$-tiling, i.e. in any position the tile in t vertically matches the one in t' in this position.

$$\mathit{match}(r, r') := \mathtt{AG}^*_{=1}\Big(\bigwedge_{(t,t') \notin V} r \wedge t \to \mathtt{AG}_{=0}(r' \to \neg t')\Big)$$

Note that the right part of the implication in this formula asserts that, if a state (ℓ, η) is contained in r, then any state (ℓ', η), i.e. one with the same clock value, is such that the locations ℓ and ℓ' are tiles matching vertically.

▶ **Lemma 13.** *Let $R = t^R_0, \dots, t^R_{2^n-1}$ and $R' = t^{R'}_0, \dots, t^{R'}_{2^n-1}$ be two rows. We have $(t_0, 0) \in [\![\mathit{match}(r, r')]\!]_{[r \mapsto R, r' \mapsto R']}$ iff $(t^R_i, t^{R'}_i) \in V$ for all $i \in [2^n]$, i.e. R' matches vertically onto R.*

We can then put this all together as follows.

$$\varphi_{\mathcal{W},n} := \exists^{\mathsf{row}} r_0.\mathsf{init}(r_0) \wedge \Big(\big(\mathtt{rec}\, \mathcal{F}(r).\mathit{final}(r) \vee \exists^{\mathsf{row}} r'.\mathit{match}(r, r') \wedge \mathcal{F}(r')\big)(r_0)\Big) \tag{1}$$

▶ **Theorem 14.** *The model checking problem for* trTRCTL *over TA is* EXPSPACE*-hard.*

Proof. By reduction from ExpTiling. From given $\mathcal{W} = (T, H, V, t_I, t_F)$ and unary $n \in \mathbb{N}$ we construct $\mathcal{A}_{\mathcal{W}}$ and $\varphi_{\mathcal{W},n}$ as described above. It is not hard to see that this can be done in time polynomial in $|W|$ and n, as the clock constraints of the form $\mathtt{x} = 2^n - 1$ etc. can be written in binary. This is where unary encoding of the parameter n in ExpTiling is needed.

Moreover, $\varphi_{\mathcal{W},n}$ is easily seen to be tail recursive as each first-order fixpoint variable – the $\mathcal{F}$ that is visible in (1) and the two $\mathcal{G}$'s that are implicitly present in the definition of the operator $\exists^{\mathsf{row}}$ – only occurs once in its corresponding body. Note that other variables, like those occurring in the macros $\mathtt{EF}^*_{=1}$ etc., are propositional only. They also occur tail recursively only but this is indeed not required for falling into the fragment trTRCTL.

At last, it remains to argue that the reduction is correct. Indeed, by Lemmas 9–13 we have $\mathcal{T}_{\mathcal{W}}, (t_0, 0) \models \varphi_{\mathcal{W},n}$ iff there is some $m \geq 1$ and a sequence of rows $R_0, \dots, R_{m-1}$ s.t. R_0 is initial, R_{m-1} is final, and R_{i+1} matches vertically onto R_i for all $i = 0, \dots, m-2$. In other words, there is a valid $\mathcal{W}$ tiling for the $[2^n] \times [m]$-corridor. ◀

5 Conclusion & Further Work

We have introduced trTRCTL, the tail-recursive fragment of TRCTL, and shown that its model-checking problem is EXPSPACE-complete. This reinforces the observation made in [10] that the complexity of TRCTL is dominated by the higher-order effects, since the lower bounds are achieved using one clock only. Hence, adding real time to RecCTL simply adds one exponential. Restricting the way recursion works reduces the complexity, while the number of clocks has no impact beyond the first one. This is notably different for TCTL [19].

We have extablished EXPSPACE hardness for the combined complexity of the trTRCTL model checking problem. However, we conjecture that the hardness result already holds for the data complexity.

Given that we now have a sound understanding of the theoretical constraints w.r.t. TRCTL and its derivatives, further research should be focused on practical applications or adding expressive power. The first aspect concerns trTRCTL in particular, as the restriction to tail recursive definitions opens up techniques like local model checking, cf. [9]. The second aspect may include making even more aspects of clock values visible to the logic, for example via so-called diagonal constraints. cf. [8].

References

1 L. Aceto and F. Laroussinie. Is your model checker on time? On the complexity of model checking for timed modal logics. *J. Log. Algebraic Methods Program*, 52-53:7–51, 2002.

2 R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. 5th Ann. IEEE Symp. on Logic in Computer Science, LICS'90*, pages 414–427. IEEE Computer Society Press, 1990. doi:10.1109/LICS.1990.113766.

3 R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

4 R. Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *Formal Methods for the Design of Real-Time Systems: Revised Lectures of the International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, pages 1–24. Springer, 2004. doi:10.1007/978-3-540-30080-9_1.

5 R. Axelsson, M. Lange, and R. Somla. The complexity of model checking higher-order fixpoint logic. *Log. Meth. in Comp. Sci.*, 3:1–33, 2007. doi:10.2168/LMCS-3(2:7)2007.

6 C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

7 P. Bouyer. Timed automata. In *Handbook of Automata Theory*, pages 1261–1294. European Mathematical Society Publishing House, 2021. doi:10.4171/Automata-2/12.

8 P. Bouyer, F. Laroussinie, N. Markey, J. Ouaknine, and J. Worrell. Timed temporal logics. In *Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, volume 10460 of *LNCS*, pages 211–230. Springer, 2017. doi:10.1007/978-3-319-63121-9_11.

9 F. Bruse, J. Kreiker, M. Lange, and M. Sälzer. Local higher-order fixpoint iteration. In *Proc. 11th Int. Symp. on Games, Automata, Logics, and Formal Verification, GandALF'20*, volume 326 of *EPTCS*, pages 97–113, 2020. doi:10.4204/EPTCS.326.7.

10 F. Bruse and M. Lange. Model checking timed recursive CTL. Submitted to Inf. and Comp.

11 F. Bruse and M. Lange. Temporal logic with recursion. In *Proc. 27th Int. Symp. on Temporal Representation and Reasoning, TIME'20*, volume 178 of *LIPIcs*, pages 6:1–6:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.TIME.2020.6.

12 F. Bruse and M. Lange. Model checking timed recursive CTL. In *Proc. 28th Int. Symp. on Temporal Representation and Reasoning, TIME'21*, volume 206 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

13 F. Bruse, M. Lange, and E. Lozes. Space-efficient fragments of higher-order fixpoint logic. In M. Hague and I. Potapov, editors, *Proc. 11th Int. Workshop on Reachability Problems, 2017, London, UK*, volume 10506 of *LNCS*, pages 26–41. Springer, 2017. doi:10.1007/978-3-319-67089-8_3.

14 S. Demri, V. Goranko, and M. Lange. *Temporal Logics in Computer Science*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016. doi:10.1017/CBO9781139236119.

15 E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982. doi:10.1016/0167-6423(83)90017-5.

16 T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

17 D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

18 M. Lange. Specifying program properties using modal fixpoint logics: A survey of results. In *Proc. 8th Indian Conf. on Logic and Its Applications, ICLA'19*, volume 11600 of *LNCS*, pages 42–51. Springer, 2019.

19 F. Laroussinie, N. Markey, and P. Schnoebelen. Model checking timed automata with one or two clocks. In *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04*, volume 3170 of *LNCS*, pages 387–401. Springer, 2004. doi:10.1007/978-3-540-28644-8_25.

20 W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

21 A. P. Sistla, E. M. Clarke, N. Francez, and A. R. Meyer. Can message buffers be axiomatized in linear temporal logic? *Information and Control*, 63(1/2):88–112, 1984.

22 P. van Emde Boas. The convenience of tilings. In *Complexity, Logic, and Recursion Theory*, volume 187 of *Lecture notes in pure and applied mathematics*, pages 331–363. Marcel Dekker, Inc., 1997.

23 M. Y. Vardi. From Church and Prior to PSL. In *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *LNCS*, pages 150–171. Springer, 2008.

24 M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04*, volume 3170 of *LNCS*, pages 512–528. Springer, 2004. doi:10.1007/978-3-540-28644-8_33.