Real-Time Higher-Order Recursion Schemes

Eric Alsmann
University of Kassel, Germany

Florian Bruse
University of Kassel, Germany

Abstract

Higher-Order Recursion Schemes (HORS) have long been studied as a tool to model functional programs. Model-checking the tree generated by a HORS of order k against a parity automaton is known to be k-EXPTIME-complete. This paper introduces timed HORS, a real-time version of HORS in the sense of Alur/Dill'90, to be model-checked against a pair of a parity automaton and a timed automaton. We show that adding dense linear time to the notion of recursion schemes adds one exponential to the cost of model-checking, i.e., model-checking a timed HORS of order k can be done in $(k+1)$-EXPTIME. This is shown by an adaption of the region-graph construction known from the model-checking of timed CTL. We also obtain a hardness result for $k = 1$, but we strongly conjecture that it holds for all k. This result is obtained by encoding runs of 2-EXPTIME Turing machines into the trees generated by timed HORS.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models; Theory of computation → Tree languages; Theory of computation → Automata over infinite objects

Keywords and phrases Timed Automata, Higher-Order Recursion Schemes, Tree Automata

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.16

1 Introduction

Higher-Order Recursion Schemes (HORS) are a well-studied framework in the context of infinite-state verification [13, 16]. A HORS is a higher-order grammar that generates a tree, e.g., the syntax tree of a functional program. The question of HORS model-checking is to decide whether a given alternating parity tree-automaton (APT) accepts the tree generated by a given HORS. This is known to be decidable in k-EXPTIME for recursion schemes of order k [20, 19]. In fact, the problem is k-EXPTIME-complete. Competitive model-checkers for this problem exist [17, 6, 5], which makes this problem practically feasible even though the theoretical complexity is high.

Another, seemingly unrelated, branch of verification of complex systems concerns real-time systems. These are systems that model time not in the discrete, transition-based fashion known from e.g., the modal μ-calculus, but rather via dense linear time. Typical verification questions then concern not the possibility of a given action, but, for example, whether the action can happen in a given timeframe as in "any request is granted in at most 5 milliseconds." A standard model for real-time verification are timed automata (TA) [2], which serve as a finite representation of an infinite system that employs dense real time. Model-checking the systems generated by timed automata is well-understood for specification logics such as timed CTL, a real-time version of the well-known temporal logic CTL [3, 4].

A recent introduction to the world of real-time verification is Timed Recursive CTL [10], which extends the specification power of timed CTL to higher-order properties, including non-regular ones. The consequence of adding such an amount of expressive power is the increase of the complexity of the model-checking problem to 2-EXPTIME.

In this paper, we follow a different approach at real-time higher-order verification, namely by making the *system* to be verified both real-time and higher-order. This is done by extending the notion of HORS to that of timed HORS, i.e., HORS annotated by real-time

information. The trees generated by timed HORS are to be model-checked simultaneously against an APT and a TA, both of which are loosely synchronized to the other. To our knowledge, this is the first introduction of real-time verification to the world of HORS.

The paper both introduces the concept of timed HORS, the respective model-checking problem, and studies the complexity of said problem. We show that adding real-time expressions to HORS increases the complexity of the model-checking problem by one exponential. The upper bound is shown for HORS of all orders, while the lower bound is shown only for order-1 HORS, due to space considerations. We are confident that the hardness result can be established for all orders due to parallels between the model-checking problems for HORS and HFL, the underlying theory of timed recursive CTL [18].

The structure of the paper is as follows: In Sect. 2, we recall the notions of HORS, their model-checking problem, as well as timed automata. In Sect. 3, we introduce timed HORS and the associated model-checking problem, supported by an example. In Sect. 4, we show that the model-checking problem for order-k timed HORS can be solved in $(k+1)$-fold exponential time, using an exponential reduction to the model-checking problem for untimed HORS. In Sect. 5, we establish a matching lower bound for $k = 1$, and we conclude with some remarks on further research in Sect. 6.

2 Preliminaries

We write $[n]$ for the set $\{1, \dots, n\}$. Let 2_0^n denote n if $k = 0$ and let 2_{k+1}^n denote $2^{2_k^n}$. We use notation of the form $(t_1, \_, t_3)$ with _ indicating that the value in question is not important, but exists.

2.1 Trees, Games, and Automata

A tree is a finite, left-closed set $T \subseteq \mathbb{N}^*$, i.e., for all $vi \in T$, we have $v \in T$ and, moreover, $vi - 1 \in T$ if $i > 0$. A *tree alphabet* is a finite, nonempty set Σ and a function $ar\colon \Sigma \to \mathbb{N}$ indicating the arity of each symbol. We write Σ_i for the set of symbols of arity i in Σ. A Σ-tree is a pair (T, l) with T a tree and $l\colon T \mapsto \Sigma$ the labeling function such that each $v \in T$ has exactly $ar(l(v))$ successors. We often identify a tree with its labeling function.

A *parity game* $\mathcal{G} = (V, V_\exists, E, v_0, \Omega)$ is a game between $\exists$ and $\forall$. Here, (V, E) is a directed graph, $V_\exists \subseteq V$ is the set of nodes owned by $\exists$, $v_0 \in V$ is the starting position, and $\Omega\colon V \to \mathbb{N}$ is the priority function with finite codomain. We write $V_\forall$ for $V \setminus V_\exists$.

A *play* of $\mathcal{G}$ is a sequence of nodes in V, connected by E and starting in v_0. A finite play is extended by the player who owns the last node in the play by picking a successor of this node. If this is not possible, the player loses the game. A *maximal* play is either infinite or cannot be extended further. $\exists$ wins an infinite play $v_0, \dots$ if the maximal number that occurs infinitely often in the sequence $\Omega(v_0), \dots$ is even, $\forall$ wins if it is odd. Positional strategies are defined as usual. A player wins a parity game iff they have a positional strategy for it. Computing who wins a parity game can be done in time $\mathcal{O}(p \cdot |E| \cdot |V|^{\lfloor \frac{p}{2} \rfloor})$ for a game with at most p priorities [15].

Let S be a set. By $\mathcal{B}^+(S)$ we denote the set of *positive Boolean expressions* over S, derived from the grammar $\varphi \coloneqq s \mid \bot \mid \top \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$ with $s \in S$.

An *alternating parity tree-automaton* (APT) is a $\mathcal{P} = (Q, \Sigma, \delta, q_I, \Lambda)$ where Q is a finite, nonempty set of *states*, Σ is a tree alphabet containing a special nullary symbol ω, $\delta = \bigcup_{i \le n} \delta^i$ is the *transition function* with $\delta^i\colon Q \times \Sigma_i \to \mathcal{B}^+(Q^i)$ and n being the maximal arity that occurs in Σ We write $\delta(q, l(v))$ for $\delta^i(q, l(v))$ if $ar(l(v)) = i$. Finally, $q_I \in Q$ is the *starting state* and $\Lambda\colon Q \to \mathbb{N} \setminus \{0\}$ is the *priority function*, which does not assign value 0 to any state unless explicitly stated otherwise. The *size* of an APT is the number of its states.

A *run* of an APT $\mathcal{P}$ on some Σ-tree T (for matching Σ) is a parity game $G(\mathcal{P}, T)$ called the *acceptance game*. It has positions from $T \times (Q \cup \bigcup_{i \leq n} \mathcal{B}^+(Q^i))$ where n is the maximal arity in Σ. Its starting position is (ϵ, q_I). In a position of the form (v, q), the unique successor is $(v, \delta(q, l(v)))$. Positions of the forms $(v, \varphi_1 \vee \varphi_2)$ and $(v, \varphi_1 \wedge \varphi_2)$ have two successors, namely (q, φ_1) and (q, φ_2). A position of the form $(v, (q_0, \dots, q_{i-1}))$ has i successors $(v0, q_0), \dots, (vi-1, q_{i-1})$, a position of the form $(v, \top)$ or $(v, \bot)$ has no successors. Positions of the forms $(v, \top)$ and $(v, \varphi_1 \wedge \varphi_2)$ and $(v, (q_0, \dots, q_{i-1}))$ belong to $\forall$, all other positions belong to $\exists$. Finally, $\Omega(v, q) = \Lambda(q)$; for all other positions we have $\Omega(v, \varphi) = 1$. $\mathcal{P}$ *accepts* a tree T if $\exists$ wins $G(\mathcal{P}, T)$.

2.2 Higher-Order Recursion Schemes

The following is quite terse, see e.g., [20] for more details.

The set of *types* is defined inductively via $\tau \coloneqq \bullet \mid \tau \to \tau$ where $\bullet$ is the type of trees, $\bullet \to \bullet$ is the type of functions that map trees to trees, etc. The *order* of a type is defined as $ord(\bullet) = 0$ and $ord(\tau_1 \to \tau_2) = \max\{1 + ord(\tau_1), ord(\tau_2)\}$. We write $\tau^i \to \tau'$ to denote the type $\tau \to \tau \to \dots \to \tau \to \tau'$ with i repetitions of τ.

Fix a tree alphabet Σ. Let $\mathcal{V} = \{x, y \dots\}$ be a set of typed *variables*. We write $x \colon \tau$ to denote that x has type τ. The set of $\mathcal{V}$-*terms* over Σ is defined inductively: for each i-ary symbol $a \in \Sigma$, the *tree constructor* a is a term of type $\bullet^i \to \bullet$, and a variable $x \in \mathcal{V}$ of type τ is a term of type τ. Given terms t_1, t_2 of types $\tau_1 \to \tau_2$ and τ_1, $(t_1\ t_2)$ is a term of type τ_2. We write $t \colon \tau$ to denote the type of a term. All terms are *subterms* of themselves; moreover t_1 and t_2 are subterms of $(t_1\ t_2)$. We use standard conventions to reduce the number of parentheses, i.e., application associates to the left. The *order* of a subterm is the order of its type; the order of a term is the maximal order of any of its subterms. By $t[t_1/x_1, \dots, t_n/x_n]$ we denote the simultaneous capture-avoiding substitution of the t_i for all occurrences of the x_i in t. Here, we assume that the types of the term and the variable it is substituted for do match.

A higher-order recursion scheme (HORS) is a $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ such that

- Σ is a tree alphabet,
- $\mathcal{N}$ is a set of typed nonterminals; we write $N \colon \tau$ to denote that N has type τ,
- $\mathcal{R}$ is a map from $\mathcal{N}$ to the set of terms such that if N has type $\tau_1 \to \dots \tau_n \to \bullet$, then $\mathcal{R}(N)$ is an $\{x_1, \dots, x_n\} \cup \mathcal{N}$-term of type $\bullet$ where the variables $x_1, \dots, x_n$ are unique to N, and
- $S \colon \bullet \in \mathcal{N}$ is the starting symbol.

The order of a recursion scheme is the maximal order of the type of any of its nonterminals. The size $|\mathcal{G}|$ of a HORS $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ is defined as the number of distinct subterms on the right-hand-sides in $\mathcal{R}$.

Define a rewriting relation $\to_{\mathcal{G}}$ via

- $N\ t_1 \cdots t_n \to_{\mathcal{G}} \mathcal{R}(N)[t_1/x_1, \dots, t_n/x_n]$ if $t_i \colon \tau_i$ for all i and $N \colon \tau_1 \to \dots \to \tau_n \to \bullet$,
- $a\ t_1 \cdots t_n \to_{\mathcal{G}} a\ t'_1 \cdots t'_n$ if $a \colon \bullet^n \to \bullet$ and $t_i \to_{\mathcal{G}} t'_i$ for all i.

Note that this defines $\mathcal{N}$-terms. For a given $\mathcal{N}$-term t, we define the tree t^ω generated by it as follows: $(a\ t_1 \cdots t_n)^\omega$ is $a\ t_1^\omega \cdots t_n^\omega$, and $(N\ t_1 \cdots t_n)^\omega$ is ω where $\omega \notin \Sigma$ is a nullary constructor. Let $(T_i, l_i) = t_i^\omega$ for $i \in \{1, 2\}$. We write $t_1^\omega \sqsubseteq t_2^\omega$ if $T_1 \subseteq T_2$ and, for all $v \in T_1$, either $l_1(v) = \omega$ or $l_1(v) = l_2(v)$. Write $T(\mathcal{G})$ for the tree generated by a recursion scheme $\mathcal{G}$, defined via $\bigsqcup\{t^\omega \mid S \to_{\mathcal{G}} t\}$. This tree is well-defined due to confluence of the lambda-calculus [14]. By abuse of notation, we also write $T(t)$ for the tree generated by a closed term $t \colon \bullet$. We assume for the rest of the paper that all HORS in question generate trees that do not contain ω in order to ease notation.

The problem of model-checking higher-order recursion schemes is now the following: given a HORS $\mathcal{G}$ over Σ and an APT $\mathcal{P}$ over $\Sigma \cup \{\omega\}$, does $\mathcal{P}$ accept $T_{\mathcal{G}}$? This problem is known to be complete for k-fold exponential time for schemes of order k [20].

2.3 Timed Automata

Let $\mathcal{X} = \{\mathtt{x}, \mathtt{y}, \dots\}$ be a set of $\mathbb{R}^{\geq 0}$-valued variables, called *clocks*. $CC(\mathcal{X})$ is the set of *clock constraints* over $\mathcal{X}$, defined as conjunctive formulas over $\top$ and $\mathtt{x} \oplus c$ for $\mathtt{x} \in \mathcal{X}$ and $c \in \mathbb{N}$, where $\oplus \in \{\leq, <, >, \geq\}$. We write $\mathtt{x} \in [c, c']$ for the constraint $\mathtt{x} \geq c \wedge \mathtt{x} \leq c'$, and similarly for open interval bounds. Clock constraints are denoted by χ, χ' etc.

A clock evaluation is a mapping $\eta \colon \mathcal{X} \to \mathbb{R}^{\geq 0}$; it satisfies a clock constraint if

- $\eta \models \top$ always,
- $\eta \models \mathtt{x} \oplus c$ iff $\eta(\mathtt{x}) \oplus c$,
- $\eta \models \varphi_1 \wedge \varphi_2$ iff $\eta \models \varphi_1$ and $\eta \models \varphi_2$.

For a clock evaluation η and $d \in \mathbb{R}^{\geq 0}$, we write $\eta{+}d$ for the clock evaluation defined via $(\eta{+}d)(\mathtt{x}) = \eta(\mathtt{x}) + d$ for all $\mathtt{x} \in \mathcal{X}$. For $R \subseteq \mathcal{X}$, $\eta|_R$ is the clock evaluation defined via $\eta|_R(\mathtt{x}) = \eta(\mathtt{x})$ if $x \notin R$ and $\eta|_R(\mathtt{x}) = 0$ if $\mathtt{x} \in R$. For singleton sets $\{\mathtt{x}\}$, we write $\eta|_x$ for $\eta|_{\{x\}}$.

Let *Prop* be a finite set of *propositions*. A *timed automaton* over clocks in $\mathcal{X}$ and with propositions in *Prop* is an $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \Delta, \lambda)$ where

- L is the set of so-called *locations* of the timed automaton, including the *initial location* ℓ_0,
- $\mathcal{X}$ is a finite set of clocks,
- $\iota \colon L \to CC(\mathcal{X})$ assigns a clock constraint called an *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 $\ell \xrightarrow{g,R} \ell'$ for $(\ell, g, R, \ell') \in \delta$. In such a transition, g is the *guard* and R are the *resets* of the transition,
- $\lambda \colon L \to 2^{Prop}$ labels each location with the propositions valid there.

The *index* of a timed automaton $\mathcal{A}$ is the largest constant that occurs in its guards or invariants. Its size is defined as

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

due to the coding of the constants in clock constraints in binary.

A TA $\mathcal{A}$ defines a so-called *timed transition system* (tTS) $(\mathcal{S}, \to, s_0, \lambda)$ over pairs of locations and clock evaluations as follows:

- The state set of the system is $\mathcal{S} = \{(\ell, \eta) \in L \times (\mathcal{X} \to \mathbb{R}^{\geq 0}) \mid \eta \models \iota(\ell)\}$.
- The initial state s_0 is (ℓ_0, η_0) where $\eta_0(\mathtt{x}) = 0$ for all $\mathtt{x} \in \mathcal{X}$.
- *Delay transitions* keep the location but let time flow: for any $(\ell, \eta) \in \mathcal{S}$ and $d \in \mathbb{R}^{\geq 0}$ we have a transition $(\ell, \eta) \xrightarrow{d} (\ell, \eta + d)$ if $\eta + d' \models \iota(\ell)$ for all $0 \leq d' \leq d$.
- *Discrete transitions* are derived from Δ: For all $(\ell, \eta) \in \mathcal{S}, \ell' \in L$ and $R \subseteq \mathcal{X}$, we have $(\ell, \eta) \to (\ell', \eta|_R)$ if there is $g \in CC(\mathcal{X})$ with $(\ell, g, R, \ell') \in \Delta$, $\eta \models g$ and, finally, $\eta|_R \models \iota(\ell')$.
- The propositional labeling λ feeds through the labeling of a location, i.e., $\lambda(\ell, \eta) = \lambda(\ell)$.
- Moreover, we consider all states labeled by the clock constraints that hold there, i.e., $(\ell, \eta) \models \chi$ iff $\eta \models \chi$.

Note that the system defined by a TA is generally neither finite nor countable, both due to the uncountable state set and the infinitary labeling of states by arbitrary clock constraints. Also note that it contains an uncountable number of transition relations, namely the anonymous transition relation that stems from discrete transitions, as well as a transition relation d for each $d \in \mathbb{R}^{\geq 0}$, stemming from delay transitions.

We say that *time can pass* from some state (ℓ_0, η_0) by an amount of $d \in \mathbb{R}^{\geq 0}$ if there is a *Trace* in the system of the form $(\ell_0, \eta_0), (\ell_1, \eta_1), \ldots, (\ell_n, \eta_n)$ such that, (I) for each $0 \leq i < n$, either $(\ell_i, \eta_i) \rightarrow (\ell_{i+1}, \eta_{i+1})$ or $(\ell_i, \eta_i) \xrightarrow{d'} (\ell_{i+1}, \eta_{i+1})$ for some $d' \in \mathbb{R}^{\geq 0}$, and (II) the sum of the delay transitions on this path is d. A state is a *timelock* if time cannot pass from this state, neither directly nor after some discrete transitions.

3 Real-Time Recursion Schemes

Let Σ be a tree alphabet. Its *timed version* is $\Sigma^t = \Sigma \times \mathcal{J}$ for $\mathcal{J}$ a finite set of intervals with natural bounds plus infinity, where an element (a, J) is written a_J. Hence, symbols in Σ^t are of the form $a_{[2,\infty)}$ or $b_{(3,5]}$ or $c_{[2,2]}$.

A *timed HORS* is a recursion scheme over a timed alphabet Σ^t. Hence, a timed HORS is a recursion scheme over a tree alphabet where the terminals, or tree constructors, are each labeled with an interval. The size of a timed HORS $\mathcal{G}$ is the size of its untimed version (obtained via the mapping $a_J \mapsto a$) times the logarithm of the largest finite interval bound mentioned by $\mathcal{G}$.

Since a timed HORS generates a tree just as an untimed HORS, this tree can be model-checked against an APT, either by simply ignoring the interval annotations (again, via the mapping $a_J \mapsto a$), or by treating Σ^t as an ordinary tree alphabet. However, the purpose of a timed HORS is to be validated against a pair of a (timed) APT and a TA, which are loosely synchronized. Before we give an intuition, we define timed APT.

Let $\mathcal{X}$ be a set of clocks and let $\mathcal{A}$ be a TA over $\mathcal{X}$ with propositions in Q. Let Ξ be a finite set of clock constraints over $\mathcal{X}$. A *timed Ξ-APT* over Σ^ω is a $(Q, \Sigma, \delta, q_I, \Lambda)$ where Q, q_I, Λ are as for ordinary APT and δ is the union of the $\delta^i \colon Q \times 2^\Xi \times \Sigma_i \to \mathcal{B}^+(Q^i)$, i.e., the transition function consumes a state, an untimed symbol from Σ, and a subset of Ξ. Note that the state set of the timed APT is exactly the set of propositions used in the TA. The size of a timed APT is $|Q| \cdot 2^{|\Xi|} \cdot \log k$, where k is the biggest clock constraint in Ξ.

Let $\mathcal{G}$ be a timed HORS over Σ^t, let $\mathcal{A}$ be a timed TA over clocks in $\mathcal{X}$, and let $\mathcal{P}$ be a timed Ξ-APT with states in Q with Ξ a set of clock constraints over $\mathcal{X}$. The semantics of an APT-TA pair over the tree generated by a timed HORS is explained in terms of an acceptance game. This works similarly to the acceptance game for ordinary, untimed APT, but with an extra component for the TA. Hence, a position of the game is defined by a node in the tree generated by the timed HORS, a state of the APT and a state of the tTS defined by the TA, i.e., a location of the TA and a suitable clock evaluation.

A play of the game proceeds like this: In a position as per above, first $\exists$ must let time flow in the tTS by an amount dictated by the labeling a_J of the current tree node. Moreover, while letting time flow, she may visit only locations of the TA whose propositional labeling includes the current state of the APT. If she cannot let time flow like this, she looses the acceptance game. Once $\exists$ has let time flow, the transition function of the timed APT is consulted. This function consumes the current state of the APT, the labeling of the current tree node, and, additionally, the clock constraints from Ξ valid in the state of the tTS defined by the TA. The transition function is then played out out as for ordinary APT. Hence, the APT and the TA are synchronized in the sense that time flow in the tTS defined by the TA is restricted by the current state of the APT, while the behavior of the APT, in turn, depends on the clock constraints valid in the current state of the tTS.

Formally, the semantics of a Ξ-APT $\mathcal{P}$ and its *companion TA* $\mathcal{A}$ (over a suitable set of clocks) over a timed HORS $\mathcal{G}$ is defined as a parity game $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$:

- The positions of the game are of the form $(v, \varphi, (\ell, \eta), b)$ where v is a node in $T_{\mathcal{G}}$, $\varphi \in Q \cup \bigcup_i \mathcal{B}(Q^i)$ is a subformula of the transition function of the timed APT, (ℓ, η) is a state in the timed transition system defined by $\mathcal{A}$, and b is a bit, indicating whether time has flown already in this position.
- The initial position is $(\epsilon, q_I, (\ell_0, \eta_0), 0)$ where $\eta_0(\mathtt{x}) = 0$ for all $\mathtt{x} \in \mathcal{X}$.
- A position of the form $(v, q, (\ell, \eta), 0)$ is owned by $\exists$. If $l(v) = a_J$, any position of the form $(v, q, (\ell', \eta'), 1)$ is a successor position, provided (ℓ', η') can be reached in the tTS from (ℓ, η) by letting time flow for an amount within J, but by visiting *only locations where the propositional labeling includes* q.
- A position of the form $(v, q, (\ell, \eta), 1)$ is owned by $\exists$. The unique successor is the position $(v, \varphi, (\ell, \eta), 1)$ where $\varphi = \delta(q, S, a)$ with $a_J = l(v)$ and $S = \{\chi \in \Xi \mid \eta \models \chi\}$.
- A position of the form $(v, \varphi_1 \vee \varphi_2, (\ell, \eta), 1)$ is owned by $\exists$, and one of the form $(v, \varphi_1 \vee \varphi_2, (\ell, \eta), 1)$ is owned by $\forall$; its successors are $(v, \varphi_i, (\ell, \eta), 1)$ for $i \in \{1, 2\}$.
- A position of the form $(v, (q_0, \ldots, q_{k-1}), (\ell, \eta), 1)$ is owned by $\forall$. Its successors are $(vi, q_i, (\ell, \eta), 0)$ for $0 \leq i < k$.
- The priority of a position of the form $(v, q, (\ell, \eta), 0)$ is $\Lambda(q)$, and 1 for the other positions.

We say that $(\mathcal{A}, \mathcal{P})$ accepts $T(\mathcal{G})$ if $\exists$ wins the above parity game.

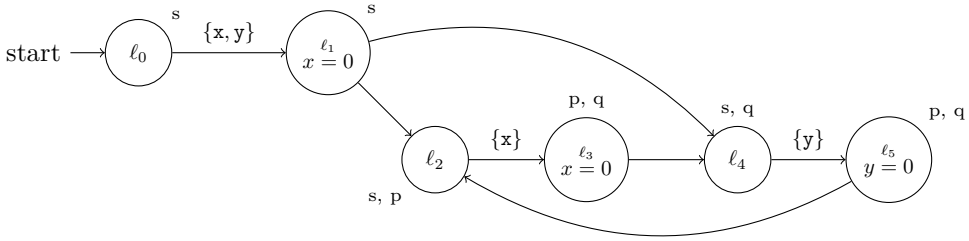
The model-checking problem for timed HORS is: Given a timed HORS $\mathcal{G}$ and an APT-TA pair $\mathcal{P}, \mathcal{A}$ as per above, decide whether $\exists$ wins $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$.

▶ **Example 1.** Consider the problem of a scheduler that schedules two processes, each of them for a variable amount of time, depending on the internal state of the scheduler. We want to verify that the scheduler is fair, i.e., that no process gets scheduled for more than k seconds in a row, for some given constant k.

Let $\{J_1, \ldots, J_n\}$ be a set of intervals denoting time slices scheduled to a process. W.l.o.g. assume that these are all unit intervals, i.e., of the form $[m, m]$ for some natural number m. Let Σ be a tree alphabet with a binary symbol a, unary symbols b, b' representing the two processes, and a nullary symbol c. We model the problem by a timed HORS with rules $S \mapsto S_1\ c$ and

$$S_1\ (x\colon \bullet) \mapsto a_{[0,0]}\ x\ \left(S_2\ \left(b^1_{J_1}\ x\right)\right) \qquad \ldots \qquad S_n\ (x\colon \bullet) \mapsto a_{[0,0]}\ x\ \left(S_1\ \left(b^n_{J_n}\ x\right)\right)$$

where the b^i are each either b or b'. Let $\mathcal{A}$ be the TA defined by the following picture:



The sets on the edges denote resets, while the equalities in locations are invariants.

Finally, let $\mathcal{P}$ be the $\{\mathtt{x} \leq k, \mathtt{y} \leq k\}$-APT defined by $(\{s, p, q\}, \Sigma, \delta, s, \lambda)$ with λ being constant 2 and δ being defined by

$$\delta(s, \_, a) = (s, s) \qquad \delta(\_, S, b) = (q)$$
$$\delta(\_, \overline{S}, \_) = \bot \qquad \delta(\_, S, b') = (p) \qquad \delta(\_, S, c) = \top$$

where $S = \{\mathtt{x} \leq k, \mathtt{y} \leq k\}$ and $\overline{S}$ stands for any set of constraints that is not S.

The timed HORS in the above example generates a tree with the root and the rightmost branch labeled by $a_{[0,0]}$, while the left branches of these nodes are increasingly longer sequences of b and b', ending in c. These encode increasingly longer scheduling decisions, encoded in reverse order. The APT traverses the rightmost branch in state s, while $\exists$ necessarily keeps the TA in location ℓ_0. Upon reading the first b or b', the APT moves to state q or p, respectively, and, hence, the TA follows by switching to locations ℓ_4, resp. ℓ_2. Switching between the two resets exactly one of the clocks. If the APT encounters the TA in a clock evaluation where one of the clocks exceeds k, the automaton rejects, and it accepts any finite branch if it reaches the final c with none of the clocks having excessive value. The infinite branch of the tree containing the as accepts since $\lambda(s) = 2$.

4 Upper Bounds for Model-Checking

We show the model-checking problem for timed HORS to be in $(k+1)$-EXPTIME for order-k timed HORS. This is done by an exponential reduction to the model-checking problem for untimed HORS. Hence, we reduce the input of a timed HORS, an APT, and a TA to a polynomially-sized untimed HORS and an exponential-sized APT such that the APT accepts the tree generated by the untimed HORS iff the APT-TA pair accepts the tree generated by the timed HORS.

As a preparation, we slightly alter the timed-HORS acceptance game. Let $\mathcal{A}$ be a TA, let $\mathcal{P}$ be an APT and let $\mathcal{G}$ be a timed HORS, all over matching alphabets and propositions. Let $\mathtt{z}$ be a clock not in the clock set $\mathcal{X}$ of $\mathcal{A}$. The acceptance game $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$ is in *extra-clock semantics*, if transitions from positions of the form $(v, q, (\ell, \eta), 0)$ are as follows:

- First, the game transitions to the position $(v, q, (\ell, \eta|_{\mathtt{z}}), 0)$. Let $l(v) = a_J$. Any position of the form $(v, q, (\ell', \eta'), 1)$ with $\eta'(\mathtt{z}) \in J$ is a successor position, provided it can be obtained by letting time flow by visiting *only locations where the propositional labeling includes* q.

▶ **Lemma 2.** *In $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$, $\exists$ wins from a position $(v, q, (\ell, \eta), 0)$ under extra-clock semantics iff she wins from that position under standard semantics. In particular, she wins the game from the initial position under extra-clock semantics iff she wins under standard semantics.*

We omit a formal proof since this is straightforward. The main point is that $\mathtt{z}$ is not reset in $\mathcal{A}$, and, hence, can serve as a yardstick for elapsed time.

4.1 The Region Abstraction

The region abstraction is a classical result (see e.g., [1] or [4], Def. 9.42), that maps the infinite transition system defined by a TA onto a finite transition system. It uses an equivalence relation $\simeq_m$, for $m \in \mathbb{N}$, on clock evaluations, defined as follows: $\eta \simeq_m \eta'$ iff

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

Here, $\mathit{frac}(r)$ denotes the fractional part of a real number. Clock evaluations are considered equivalent if, for each clock, (I) either both clocks have a value greater than m, or (II) they compare in the same way with respect to all integers less than m. Moreover, the passage of time will make equivalent evaluations reach the next integral value first for the same clock.

It is straightforward to verify that $\simeq_m$ is an equivalence relation for any m. An equivalence class in this relation is called a *region*. We denote the equivalence class of η under $\simeq_m$ as $[\eta]_m$, When m is clear from the context, we may omit it and simply write $[\eta]$

An important observation is that the region abstraction is also a congruence w.r.t. the winner of the timed-HORS acceptance game:

▶ **Lemma 3.** *Let $\mathcal{A}$ be a TA, $\mathcal{P}$ a Ξ-APT, $\mathcal{G}$ a timed HORS, all over matching alphabets and clocks. Let m be greater than or equal to the index of $\mathcal{A}$, any interval on a tree constructor in $\mathcal{G}$, and any clock constraint in Ξ. Let v be a node in $T(\mathcal{G})$, q a state of $\mathcal{P}$, ℓ a location in $\mathcal{A}$ and $\eta \simeq_m \eta'$ two clock evaluations. Then $\exists$ wins the acceptance game from position $(v, q, (\ell, \eta), 0)$ iff she wins the game from position $(v, q, (\ell, \eta'), 0)$.*

The proof is in App. A.1. By the lemma, the outcome of the acceptance game does not depend on the exact clock evaluation, but only on the region in question. This motivates notation of the form "$\exists$ wins from a position $(v, q, (\ell, [\eta]), 0)$", which means that $\exists$ wins for all positions $(v, q, (\ell, \eta'), 0)$ with $\eta' \in [\eta]$.

As mentioned above, given a TA $\mathcal{A}$ and m greater than or equal to the index of $\mathcal{A}$, the region abstraction induces a finite transition system that faithfully represents the transition system defined by the TA. To make this precise, we define the notion of a *successor region*:

▶ **Definition 4.** *Let $\mathcal{A}$ be a TA, and let m be greater or equal than the index of $\mathcal{A}$. Let $\simeq_m$ denote the region equivalence w.r.t. m. For each region $[\eta]_m$, the unique* successor region *is*

- *$suc([\eta]_m) = [\eta]_m$ if $\eta(\mathbf{x}) > m$ for all $\mathbf{x} \in \mathcal{X}$,*
- *$suc([\eta]_m) = [\eta']_m$ 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 $[\eta]_m \neq [\eta']_m$.*

The second term defines the successor region of $[\eta]$ to be the first region that is entered if time passes from any $\eta' \in [\eta]$, and the first term makes the successor region well-defined in regions where all clocks have values greater than m.

Let $\mathcal{A}$ be a TA and let m be greater than or equal to the index of $\mathcal{A}$. Let Ξ be a set of clock constraints over the clocks of $\mathcal{A}$. The *region graph* $\mathcal{R}^m_\Xi(\mathcal{A})$ of $\mathcal{A}$ (w.r.t. m and Ξ) is the transition system defined as follows:

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

The following is a classical result.

▶ **Proposition 5** ([1])**.** *Let $\mathcal{A}$ be a TA over n clocks with l locations and let Ξ be a set of clock constraints over the clocks in $\mathcal{A}$. Let m be greater than or equal to the index of $\mathcal{A}$. Then $\mathcal{R}^m_\Xi(\mathcal{A})$ is an (untimed) transition system of size $l \cdot 2^{\mathcal{O}n(\log n + \log m)} \cdot |\Xi|$, i.e., exponential in $|\mathcal{A}|$, and there is a trace $(\ell_0, \eta_0), \ldots, (\ell_n, \eta_n)$ in the system defined by $\mathcal{A}$ iff there is a path $(\ell_0, [\eta_0] \to [s_1] \to \ldots [s_{n'-1}] \to (\ell_n, [\eta_n])$ in $\mathcal{R}^m_\Xi(\mathcal{A})$.*

The last statement means that letting time flow in the tTS defined by $\mathcal{A}$ corresponds to following a path in $\mathcal{R}^m_\Xi(\mathcal{A})$, and vice versa.

4.2 The Reduction

We now begin the reduction. The overall idea is to replace the companion TA, and the infinite transition system defined by it, by its corresponding region graph. This untimed transition system is then incorporated into the APT via a product construction. Hence, the new APT will have states of the form $(q, \ell, [\eta])$, where q is a state of the original APT, ℓ is a location in the companion TA, and $[\eta]$ is a region in its region graph. The transition function is then modified to simulate the passage of time from positions of the form $(v, q, (\ell, \eta), 0)$ using extra-clock semantics. However, direct simulation of the passage of time is tricky. In particular, it cannot be directly incorporated into the transition function without incurring blowup by another exponential in addition to the exponential blowup caused by passage to the region graph.

The above construction is supported by changing the timed HORS into an untimed HORS. This untimed HORS defines a tree that is similar in structure to that defined by the timed one, and it also makes the interval annotations of the tree defined by the timed HORS explicit via special gadgets. These gadgets help the new APT to properly advance time in its region graph components.

Let $\mathcal{G} = (\Sigma^t, \mathcal{N}, \mathcal{R}, S)$ be a timed HORS over a timed alphabet Σ^t, together with its untimed version Σ. Let $\mathcal{J}$ be the intervals occurring on the right-hand sides of $\mathcal{R}$. Define the untimed alphabet $\Sigma^u = \Sigma \cup \{\mathsf{reset}, \mathsf{flow}\} \cup \bigcup_{J \in \mathcal{J}} \mathsf{check}^J$ where $\mathsf{reset} \colon \bullet \to \bullet$, $\mathsf{flow} \colon \bullet^2 \to \bullet$ and $\mathsf{check}^J \colon \bullet \to \bullet$ for all $J \in \mathcal{J}$. Then let $\mathcal{G}^u = (\Sigma^u, \mathcal{N} \cup \bigcup_{J \in \mathcal{J}} \{\mathtt{start}^J, \mathtt{choose}^J\}, \mathcal{R}', S)$ be an untimed HORS where $\mathcal{R}'$ is defined as via:

$$\mathtt{start}^J(x \colon \bullet) \mapsto \mathsf{reset}\,(\mathtt{choose}^J\, x) \qquad \mathtt{choose}^J(x \colon \bullet) \mapsto \mathsf{flow}\,(\mathtt{choose}^J\, x)\,(\mathsf{check}^J\, x)$$

and the right-hand side for a symbol in N is defined as $\mathcal{R}'(N) = \widehat{\mathcal{R}(N)}$ where $(a_J\ \widehat{t_1 \cdots t_n}) = \mathtt{start}^J(a\ \hat{t_1} \cdots \hat{t_n})$, $\widehat{t_1\ t_2} = \hat{t_1}\ \hat{t_2}$ if $t_1 \neq a_J$, $\hat{x} = x$ for a variable x, and $\hat{N'} = N'$ for $N' \in \mathcal{N}$.

Hence, a tree of the form $a_J\ t_1\ \cdots t_n$ is replaced by a tree whose root is labeled by reset. The unique subtree below this root is an infinite tree where all nodes on the leftmost branch are labeled by flow, and the right son of each of the nodes labeled by flow is a tree of the form $\mathsf{check}^J\ (a\ \hat{t_1}\ \cdots \hat{t_n})$. Hence, there are infinitely many copies of the latter subtree, each reached by following the left son of the nodes labeled by flow for a finite number of times, and then branching to the right once.

The intuition is that the sequence of nodes labeled by flow will help the yet-to-be-defined untimed APT to simulate the flow of time in the tTS defined by the TA by following a suitable path in its region graph component, i.e., one that advances the region component from one region to its successor region. The infinite sequence of nodes labeled by flow allows the APT to follow paths of arbitrary length in this fashion. Conversely, the passage of time is an an atomic action in the semantics of the (timed) APT-TA pair, which must be broken up into its individual steps to avoid exponential blowup.

Let $\mathcal{G}$ be as before and let $\mathcal{P} = (Q, \Sigma, \delta, q_I, \Lambda)$ be a timed Ξ-APT over Σ^t, let $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \Delta, \lambda)$ be a companion TA for it. Let $Q' = Q \cup \{q' \mid q \in Q\}$, let m be the smallest integer at least as big as the index of $\mathcal{A}$, each finite interval bound in $\mathcal{J}$ and each clock constraint in Ξ. Let Ξ' be the union of Ξ and $\{\mathtt{z} \in J \mid J \in \mathcal{J}\}$, where $\mathtt{z} \notin \mathcal{X}$.

Define an untimed APT $\mathcal{P}_\mathcal{A} = (Q'', \Sigma^u, \delta', q_I', \Lambda')$ via

- $Q'' = \{(q'', \ell, [\eta]) \mid q'' \in Q', (\ell, [\eta]) \text{ is a state in } \mathcal{R}^m_\Xi(\mathcal{A})\}$
- the initial state is $q_I'' = (q_I, \ell_0, [\eta_0])$,
- $\Lambda'(q'', \_, \_) = \Lambda(q'')$ if $q'' \in Q$, otherwise $\Lambda'(q'', \_, \_) = 1$

and δ' is defined as

$$\begin{aligned}
\delta'((q,\ell,[\eta]),\mathsf{reset}) &= (q',\ell,[\eta|_{\mathsf{z}}]) \text{ if } q \in Q\\
\delta'((q',\ell,[\eta]),\mathsf{flow}) &= ((q',\ell',[\eta']),(q',\ell,[\eta])) \text{ if } (\ell,[\eta]) \to (\ell',[\eta']) \text{ in } \mathcal{R}^m_\Xi(\mathcal{A}) \text{ and } q \in Q\\
\delta'((q',\ell,[\eta]),\mathsf{check}^J) &= \bot \text{ if } \eta(\mathsf{z}) \notin J\\
\delta'((q',\ell,[\eta]),\mathsf{check}^J) &= (q,\ell,[\eta]) \text{ if } \eta(\mathsf{z}) \in J\\
\delta'((q,\ell,[\eta]),a) &= \varphi' \text{ if } \delta(q,S,a) = \varphi \text{ and } S = \{\chi \in \Xi \mid \eta \models \chi\}
\end{aligned}$$

where φ' is obtained from φ by replacing all occurrences of $q \in Q$ by $(q,\ell,[\eta])$. All unlisted transitions can be assumed to be $\bot$. The intuition is that the copies q' of states in Q serve for the sequence of flow nodes and have priority 1 to avoid staying on the sequence indefinitely. Branching leftward at a flow node advances time. The following is immediate from Prop. 5.

▶ **Observation 6.** *The size of $\mathcal{P}_\mathcal{A}$ is exponential in that of $\mathcal{P}$ and $\mathcal{A}$.*

We now show that $\mathcal{P}_\mathcal{A}$ simulates the APT-TA pair of $\mathcal{P}$ and $\mathcal{A}$ faithfully.

▶ **Lemma 7.** *Let $\mathcal{G},\mathcal{A}$ and $\mathcal{P}$ be a timed HORS, a TA and an APT over matching alphabets. $\mathcal{P}_\mathcal{A}$ accepts the tree generated by $\mathcal{G}^u$ iff the APT-TA pair accepts $T_\mathcal{G}$.*

The proof is in App. A.1.

▶ **Theorem 8.** *The model-checking problem for order-k timed HORS is in $(k+1)$-EXPTIME.*

Proof. By Lemma 7, the model-checking problem for a timed APT $\mathcal{P}$ and a TA $\mathcal{A}$ over a timed HORS $\mathcal{G}$ can be reduced to the untimed model-checking problem of $\mathcal{P}_\mathcal{A}$ over $\mathcal{G}^u$. By Obs. 6, the size of $\mathcal{P}_\mathcal{A}$ is exponential in that of $\mathcal{P}$ and $\mathcal{A}$. Since the model-checking problem for order-k HORS can be solved in k-fold exponential time [20], the result follows. ◀

5 Matching Lower Bounds

We now show that the model-checking problem for order-1 timed HORS is 2-EXPTIME hard. The proof reuses a construction from [10], adapted to the situation with timed HORS. We restrict ourselves to the order-1 case due to space considerations, but we conjecture that the proof can be extended to arbitrary order in the same fashion as in [10].

5.1 Encoding 2-EXPTIME Turing Machines

We use an idea from [12] to encode the word problem for deterministic 2-EXPTIME Turing machines, namely by encoding an accepting run of the machine into a square table of 2-fold exponential size. The rows of the table are the configurations of the run, in order. A row lists the tape contents of the machine and the position and state of the head.

The following is obtained by standard reductions: the machine never moves its head to the left of the starting position and it accepts by moving to a unique accepting state at the starting position and on empty tape. Moreover, instead of halting, it repeats this configuration ad infinitum, and it enters this configuration in time at most $2^{2^n} - 2$ on input of size n instead of merely time less than $2^{2^{p(n)}}$ for some polynomial p. Moreover, it suffices to consider the empty-word problem, i.e., the question whether the machine halts on empty input. The last assumption is easily shown to be valid, as for every machine $\mathcal{M}$ and input w satisfying all the other assumptions, there is $\mathcal{M}_w$ that first writes w on empty input and then behaves like $\mathcal{M}$.

Let $\mathcal{M} = (Q, \Sigma, \Gamma, q_I, q_{\text{acc}})$ be a DTM with state set Q, unique initial and accepting states q_I and q_{acc}, input and tape alphabets $\Sigma \subsetneq \Gamma$, with $\square \in \Gamma \setminus \Sigma$ the blank symbol and, $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ the transition function. Let $\hat{\Gamma} = \Gamma \cup (Q \times \Gamma) \cup \{\#\}$ where $\#$ is a new boundary symbol that does not occur in Γ. Define the extended transition function $\hat{\delta}\colon \hat{\Gamma}^3 \to \hat{\Gamma} \setminus \{\#\}$ with $\hat{\delta}(b_1, b_2, b_3)$ being

$$
\begin{aligned}
&(b_2, q), \text{ if } b_1 \in \Gamma \cup \{\#\}, b_2 \in \Gamma \text{ and } b_3 = (q', a) \text{ with } \delta(q', a) = (q, \_, L)\\
&(b_2, q), \text{ if } b_3 \in \Gamma \cup \{\#\}, b_2 \in \Gamma \text{ and } b_1 = (q', a) \text{ with } \delta(q', a) = (q, \_, R)\\
&(a, q), \text{ if } b_1, b_3 \in \Gamma \cup \{\#\}, b_2 = (q', a) \text{ with } \delta(q', a) = (q, b, N)\\
&a, \text{ if } b_1, b_3 \in \Gamma \cup \{\#\}, b_2 = (q', b) \text{ with } \delta(q', a) = (q, a, R) \text{ or } (q, a, L)\\
&a, \text{ if } b_2 = a, b_1 \neq (q', a') \text{ with } \delta(q', a') = (q, \_, R), b_3 \neq (q', a') \text{ with } \delta(q', a') = (q, \_, L)
\end{aligned}
$$

This encodes the contents of a cell in the table of rows in dependence on the three cells of the previous row that are directly below it, resp. directly adjacent to the cell directly below it. It is easy to see that the preimage of any $b \in \hat{\Gamma} \setminus \{\#\}$ under $\hat{\delta}$ is easy to compute, since there are only $|\hat{\Gamma}^3|$ many possible candidates.

▶ **Definition 9.** *A 2_2^n-certificate for the acceptance of $\mathcal{M}$ is a $C\colon [2_2^n] \times [2_2^n] \to \hat{\Gamma}$ where*

1. *$C(r, c) = \#$ iff $c = 0$ or $c = 2^{2^n} - 1$,*
2. *$C(0, 1) = (q_I, \square)$ and $C(0, c) = \square$ if $2 \leq c < 2^{2^n} - 1$,*
3. *if $r > 0$, $0 \neq c \neq 2^{2^n} - 1$ and $C(r, c) = b \in \hat{\Gamma}$, then there are b_1, b_2, b_3 with $\hat{\delta}(b_1, b_2, b_3) = b$ and $C(r-1, c-1) = b_1, C(r-1, c) = b_2$ and $C(r-1, c+1) = b_3$.*

Such a certificate is successful *if also $C(2^{2^n} - 1, 0) = (q_{acc}, \square)$ holds.*

Clearly, for every $\mathcal{M}$ and n, there is a unique such certificate.

▶ **Proposition 10.** *It is a 2-EXPTIME-hard problem to decide whether, given some n encoded in unary, and a DTM $\mathcal{M}$, there is a successful 2_2^n-certificate for $\mathcal{M}$ and n in the sense above.*

For a more detailed exposition of this, see [10].

5.2 Modeling of Large Numbers

In order to encode a certificate as in the previous section into the model-checking problem for timed HORS, we need to encode and manipulate numbers up to 2^{2^n} into a polynomially-sized presentation of a timed HORS, a TA and an APT. The heavy lifting for the encoding is done by the TA, via the use of clock values, while the manipulating is done using the HORS.

The encoding is done using a single clock. Let $\mathcal{A}$ be the TA with one location ℓ, one clock x, no invariants and no transitions and, hence, no guards. Hence, a state in the transition system is defined completely by its value of x. Let $\Sigma_{\text{red}} = \{\mathsf{and}, \mathsf{and3}, \mathsf{or}, \mathsf{or3}, \mathsf{neg}, \mathsf{ff}, \mathsf{mx}\}$ be a tree alphabet where ff and mx are of type $\bullet$, neg is of type $\bullet \to \bullet$, and and or are of type $\bullet^2 \to \bullet$ and $\mathsf{and3}$ and $\mathsf{or3}$ are of type $\bullet^3 \to \bullet$. Let $\Xi = \{\mathsf{x} \leq 2^n - 1\}$. We abbreviate this single constraint by $\mathtt{max}$. Let $\mathcal{P}_{\text{cnt}} = (\{q_1^{\text{yes}}, q_0^{\text{yes}}, q_1^{\text{no}}, q_0^{\text{no}}, q^\top\}, \Sigma_{\text{red}}, \delta, q_1^{\text{yes}}, \Lambda)$ be a timed APT with $\Lambda(q_1^{\text{yes}}) = \Lambda(q_1^{\text{no}}) = \Lambda(q^\top) = 1$ and $\Lambda(q_0^{\text{yes}}) = \Lambda(q_0^{\text{no}}) = 0$, with δ to be defined soon. The intuition is that $q_1^{\text{yes}}, q_0^{\text{yes}}$ verify that a given bit is set in the representation of a large number, and that $q_1^{\text{no}}, q_0^{\text{no}}$ falsify this, with the difference between the two copies of each state to become important later. The state $q^\top$ accepts any tree. We set the location ℓ in $\mathcal{A}$ to satisfy all propositions associated to states in Q.

The transition function of the APT is defined via (where $i \in \{1, 2\}$)

$$\begin{aligned}
\delta(q_i^{\text{yes}}, _, \mathsf{ff}) &= \bot & \delta(q_i^{\text{yes}}, _, \mathsf{and}) &= (q_0^{\text{yes}}, q_1^{\text{yes}}) \\
\delta(q_i^{\text{no}}, _, \mathsf{ff}) &= \top & \delta(q_i^{\text{yes}}, _, \mathsf{or}) &= (q_1^{\text{yes}}, q^\top) \vee (q^\top, q_1^{\text{yes}}) \\
\delta(q_i^{\text{yes}}, \{\mathtt{max}\}, \mathsf{mx}) &= \top & \delta(q_i^{\text{no}}, _, \mathsf{and}) &= = (q_1^{\text{no}}, q^\top) \vee (q^\top, q_1^{\text{no}}) \\
\delta(q_i^{\text{no}}, \{\mathtt{max}\}, \mathsf{mx}) &= \bot & \delta(q_i^{\text{no}}, _, \mathsf{or}) &= = (q_0^{\text{no}}, q_1^{\text{no}}) \\
\delta(q_i^{\text{yes}}, \emptyset, \mathsf{mx}) &= \bot & \delta(q_i^{\text{yes}}, _, \mathsf{and3}) &= (q_1^{\text{yes}}, q_1^{\text{yes}}, q_1^{\text{yes}}) \\
\delta(q_i^{\text{no}}, \emptyset, \mathsf{mx}) &= \top & \delta(q_i^{\text{yes}}, _, \mathsf{or3}) &= (q_1^{\text{yes}}, q^\top, q^\top) \vee (q^\top, q_1^{\text{yes}}, q^\top) \vee (q^\top, q^\top, q_1^{\text{yes}}) \\
\delta(q^\top, _, _) &= \top & \delta(q_i^{\text{no}}, _, \mathsf{and3}) &= (q_1^{\text{no}}, q^\top, q^\top) \vee (q^\top, q_1^{\text{no}}, q^\top) \vee (q^\top, q^\top, q_1^{\text{no}}) \\
& & \delta(q_i^{\text{no}}, _, \mathsf{or3}) &= (q_1^{\text{no}}, q_1^{\text{no}}, q_1^{\text{no}}).
\end{aligned}$$

The intuition for the automaton is that the states q_i^{yes} reject at ff (for "false") and that they accept at mx iff, in the companion TA, `max` is satisfied. The states q_i^{no} work in the opposite fashion. At tree constructors and, or, both state pairs behave as expected: for and, both subtrees have to be verified, while for or, only one of them needs to be verified, and opposite for q_i^{no}. Again, the pair q_i^{no} behaves in the opposite fashion, and and3 and or3 are the ternary variants of and and or. For reasons that we will explain later, at the binary tree constructors, the priority of the state going in leftward direction sometimes deviates from the default 1.

▶ **Definition 11.** *A tree over Σ_{red} encodes a number $k \in [2^{2^n}]$ if its representation in binary is $\sum_{i=0}^{2^n-1} b_i \cdot 2^i$ where $b_i = 0$ iff $\mathcal{P}_{cnt}$ accepts the root of the tree with the clock of its companion TA set to $2^n - 1 - i$.*

Note that this definition talks about runs of the APT-TA pair with the clock of the TA initialized to a specific value, not necessarily 0. The intuition is that acceptance of $\mathcal{P}_{\text{cnt}}$ for different initial values of the clock of the companion TA forms a set of bits: for each clock value in $[2^n]$, $\mathcal{P}_{\text{cnt}}$ either accepts or it does not. These bits can be thought of as the binary representation of a number in $[2_2^n - 1]$. Note that the order of the bits is reversed from what one would intuitively expect: the least significant bit is the one where x has value $2_2^n - 1$, while the most significant bit is the one where x has value 0.

It is also fairly easy to see that there are trees that encode numbers. For example, the number 0 is encoded by the tree with just one node, labeled by ff. It now is time to add incrementation and decrementation of numbers.

In the binary representation of the increment of a number, a bit is set if either it is

- set in the representation of the number, and there is an unset bit of lesser significance, or
- not set in the representation of the number, but all bits of lesser significance are set.

Since bits are set in the encoding of our numbers if the APT $\mathcal{P}_{\text{cnt}}$ accepts if the single clock of its companion TA is set to the right value, checking whether a bit of lesser significance is set in the encoding of a number, i.e., a tree, can be done by letting time flow for the right amount, and then simply checking if the APT accepts the tree. This works because bits of lesser significance are represented by clock values closer to $2_2^n - 1$.

In the following, for $a_{[1,1]}$ we write a_1, and an undecorated constructor a means $a_{[0,0]}$. Consider the following definitions, where $T(\mathtt{zero})$ clearly encodes 0:

$$
\begin{aligned}
\texttt{zero} &\mapsto \mathsf{ff}\\
\texttt{exists}\ (x\colon \bullet) &\mapsto \mathsf{or}_1\ (\texttt{exists}\ x)\ (\mathsf{and}\ \mathtt{max}\ x)\\
\texttt{all}\ (x\colon \bullet) &\mapsto \mathsf{and}_1\ (\texttt{all}\ x)\ (\mathsf{or}\ (\mathsf{neg}\ \mathtt{max})\ x)\\
\texttt{inc}\ (x\colon \bullet) &\mapsto \mathsf{or}\big(\mathsf{and}\ x\ (\texttt{exists}\ (\mathsf{neg}\ x))\big)\ \big(\mathsf{and}\ (\mathsf{neg}\ x)\ (\texttt{all}\ x)\big)\\
\texttt{dec}\ (x\colon \bullet) &\mapsto \mathsf{or}\big(\mathsf{and}\ x\ (\texttt{exists}\ x)\big)\ \big(\mathsf{and}\ (\mathsf{neg}\ x)\ (\texttt{all}\ (\mathsf{neg}\ x))\big)\\
\texttt{isZero?}\ (x\colon \bullet) &\mapsto \mathsf{and}(\mathsf{neg}\ x)\ (\texttt{all}\ \mathsf{neg}\ x)
\end{aligned}
$$

We have already seen that the tree generated by `zero` encodes the number 0. The following lemma proves that the above macros behave as their names suggest.

▶ **Lemma 12.** *Let e be a closed term of type $\bullet$ such that $T(e)$ encodes some number $k \in [2^{2^n}]$. Then*

1. *$\mathcal{P}_{cnt}$ accepts $T(\texttt{exists}\ e)$ from states q_i^{yes} with its clock set to some value $k \in [2^n]$ iff it accepts $T(e)$ from q_i^{yes} with the clock set to some integral k' with $k < k' \leq 2^n - 1$, and it accepts from states q_i^{no} iff it accepts $T(e)$ from q_i^{no} for no such integral clock value.*
2. *$\mathcal{P}_{cnt}$ accepts $T(\texttt{all}\ e)$ from states q_i^{yes} with its clock set to some value $k \in [2^n]$ iff it accepts $T(e)$ from q_i^{no} and for all integral clock values k' with $k < k' \leq 2^n - 1$, and it accepts from states q_i^{no} iff it rejects $T(e)$ from q_i^{no} for some such integral clock value.*
3. *$T(\texttt{inc}\ e)$ encodes the number $k + 1 \mod 2^{2^n}$ and $T(\texttt{dec}\ e)$ encodes the number $k - 1 \mod 2^{2^n}$.*
4. *$\mathcal{P}_{cnt}$ accepts $T(\texttt{isZero?}\ e)$ with the clock set to 0 from states q_i^{yes} iff $T(e)$ encodes 0, and from states q_i^{no} iff the tree generated by $T(e)$ encodes a number different from 0.*

The proof is in App. A.2.

5.3 The Hardness Proof

We now show that it is 2-EXPTIME-hard to decide the model-checking problem of timed HORS , i.e., to decide whether a given pair of a TA and a timed APT accepts the tree generated by an order-1 timed HORS. We reduce the following problem to the model-checking problem: given n encoded in unary and $\mathcal{M}$ a DTM, is there a successful 2_2^n-certificate for $\mathcal{M}$? By Prop. 10, this problem is 2-EXPTIME-hard.

For the remainder of this section, fix some n encoded in unary and a 2_2^n-bounded DTM. Let $\hat{\Gamma}$ be as in Def. 9. Note that the question for a successful 2_2^n-certificate asks for a table of width 2_2^n and height 2_2^n, filled with entries from $\hat{\Gamma}$. It is sufficient to check that the top left entry of the certificate is correct, and then to recursively check that the three entries directly below a given entry are as per $\hat{\delta}$.[1]

We implement this recursive verification procedure by defining an APT-TA pair and an order-1 timed HORS such that the pair accepts the HORS iff the certificate is successful. The state of the APT represents an element of $\hat{\Gamma}$. The timed HORS generates a tree in which certain nodes represent an entry of the table, and these nodes have two (indirect) subtrees that encode the row and column number of the entry, to be recognized by the TA part of the APT-TA pair as in Def. 11. In addition, such a node representing a column entry has another three direct subtrees, which encode the three table entries in the previous row with the three adjacent column numbers as per $\hat{\delta}$. The tree contains further gadgets to verify that a given symbol can occur in the entry if this is easily verified, e.g., for the leftmost and rightmost columns and the boundary symbol #. The scheme is defined as follows:

[1] This recursive procedure does not check the top right half of the table, but it is easy to see that it must be filled by blanks due to the assumptions about the DTM in question. See also [10] for more details.

$$
\begin{aligned}
S &\mapsto F\ (\texttt{dec zero})\ \texttt{zero}\\
F(r\colon\bullet)(c\colon\bullet) &\mapsto \mathsf{or}\ (\texttt{check}\ r\ c)\ (\texttt{next}\ r\ c)\\
\texttt{gtOne?}\ (x\colon\bullet) &\mapsto \mathsf{and}\ \big(\texttt{neg}\ (\texttt{isZero?}\ x)\big)\ \big(\texttt{neg}\ (\texttt{isZero?}\ (\texttt{dec}\ x))\big)\\
\texttt{ltMax?}\ (x\colon\bullet) &\mapsto \mathsf{and}\ \big(\texttt{neg}\ (\texttt{isZero?}\ (\texttt{inc}\ x))\big)\ \big(\texttt{isZero?}\ (\texttt{inc}\ (\texttt{inc}\ x))\big)\\
\texttt{check}\ (r\colon\bullet)(c\colon\bullet) &\mapsto \mathsf{or3}\ \big(\mathsf{or}\ (\texttt{isZero?}\ (\texttt{inc}\ r))\ (\texttt{isZero?}\ r)\big)\\
&\qquad\big(\mathsf{and}\ (\texttt{isZero?}\ r)\ (\texttt{isZero?}\ (\texttt{dec}\ c))\big)\\
&\qquad\big(\mathsf{and3}\ (\texttt{isZero?}\ r)\ (\texttt{gtOne?}\ x)\ (\texttt{ltMax?}\ x)\big)\\
\texttt{next}\ (r\colon\bullet)(c\colon\bullet) &\mapsto \mathsf{and}\ \Big(\mathsf{and3}\ (\texttt{neg}\ (\texttt{isZero?}\ r))\ (\texttt{neg}\ (\texttt{isZero?}\ c))\ (\texttt{ltMax?}\ c)\Big)\\
&\qquad\Big(\mathsf{and3}\ \big(F\ (\texttt{dec}\ r)\ (\texttt{dec}\ c)\big)\ \big(F\ (\texttt{dec}\ r)\ c\big)\ \big(F\ (\texttt{dec}\ r)\ (\texttt{inc}\ c)\big)\Big)
\end{aligned}
$$

Intuitively, the only properly recursive nonterminal here is F. Any subtree generated by F with arguments r and c encodes a cell of the certificate, namely that whose row number is encoded by $T(r)$ and its column number is encoded by $T(c)$. Such a tree node poses a choice to the yet-to-be-defined APT whether it accepts this tree via (the tree generated by) `next` or via `check` . The latter variant allows the APT to verify that $T(r)$ and $T(c)$ satisfy the boundary conditions of the certificate, while the former variant verifies that $T(r)$ and $T(c)$ encode the indices of an interior cell of the certificate and make the APT verify recursively the properties of the three cells below that encoded by $T(r)$ and $T(c)$.

Define $\mathcal{P}_{\mathrm{red}} = (Q, \Sigma_{\mathrm{red}}, \delta', \{q^{(q_{\mathrm{acc}},\square)}\}, \Lambda)$ as an extension of the APT $\mathcal{P}_{\mathrm{cnt}}$ from the previous section, with δ' to be defined soon, with

$$Q = \{q_1^{\mathrm{yes}}, q_0^{\mathrm{yes}}, q_1^{\mathrm{no}}, q_0^{\mathrm{no}}, q^{\top}\} \cup \{q_l^b \mid b \in \{\#, \square, (q_I, \square)\}\} \cup \{q^{(b_1,b_2,b_3)} \mid b_1, b_2, b_3 \in \hat{\Gamma}\},$$

and $\Lambda(q) = 1$ for all $q_l^b, q^{(b_1,b_2,b_3)}$ and Λ as per $\mathcal{P}_{\mathrm{cnt}}$ for $q \in \{q_1^{\mathrm{yes}}, q_0^{\mathrm{yes}}, q_1^{\mathrm{no}}, q_0^{\mathrm{no}}, q^{\top}\}$. The companion TA remains the one-state, one-clock featureless TA from $\mathcal{P}_{\mathrm{cnt}}$.

Consider the timed HORS $\mathcal{G}_{\mathrm{red}} = (\Sigma_{\mathrm{red}}, \mathcal{N}, \mathcal{R}, S)$ with $\mathcal{N}$ and $\mathcal{R}$ as given by the definitions above. The idea is that $\mathcal{P}_{\mathrm{red}}$ accepts from state q^b and with the clock of its companion TA set to 0 the tree $T(F\ r\ c)$ iff the numbers encoded by the trees $T(r)$ and $T(c)$ are m and m' such that $C(m, m') = b$ in the unique certificate for $\mathcal{M}$ and n. In particular, $\mathcal{P}_{\mathrm{red}}$ accepts $T(S)$ from its starting state $q^{(q_{\mathrm{acc}},\square)}$ iff $C(2_2^n - 1, 0) = (q_{\mathrm{acc}}, \square)$.

Towards this, consider the following definition of the transition function of $\mathcal{P}_{\mathrm{red}}$:

$$
\begin{aligned}
\delta'(q^b, \_, \mathsf{or}) &= (q_l^b, q^{\top}) \text{ if } b \in \{\#, (q_I, \square)\}\\
\delta'(q^{\square}, \_, \mathsf{or}) &= (q_l^{\square}, q^{\top}) \vee \bigvee_{\hat{\delta}(b_1,b_2,b_3)=\square} (q^{\top}, q^{(b_1,b_2,b_3)})\\
\delta'(q^b, \_, \mathsf{or}) &= \bigvee_{\hat{\delta}(b_1,b_2,b_3)=b} (q^{\top}, q^{(b_1,b_2,b_3)}) \text{ if } b \notin \{\#, \square, (q_I, \square)\}
\end{aligned}
$$

$$
\begin{aligned}
\delta'(q_l^{\#}, \_, \mathsf{or3}) &= (q_1^{\mathrm{yes}}, q^{\top}, q^{\top}) & \qquad \delta'(q_l^{(q_i,\square)}, \_, \mathsf{or3}) &= (q^{\top}, q_1^{\mathrm{yes}}, q^{\top})\\
\delta'(q_l^{\square}, \_, \mathsf{or3}) &= (q^{\top}, q^{\top}, q_1^{\mathrm{yes}}) & \qquad \delta'(q^{(b_1,b_2,b_3)}, \_, \mathsf{and}) &= (q_1^{\mathrm{yes}}, q^{(b_1,b_2,b_3)})\\
\delta'(q^{(b_1,b_2,b_3)}, \_, \mathsf{and3}) &= (q^{b_1}, q^{b_2}, q^{b_3})
\end{aligned}
$$

The remaining transitions can all be assumed to lead to $\bot$, as they will not occur anyway.

We now formalize the above intuition about the semantics of the APT.

▶ **Lemma 13.** *Let $C\colon 2_2^n \times 2_2^n \to \hat{\Gamma}$ be the unique certificate for n and $\mathcal{M}$. Let $r, c\colon \bullet$ be expressions such that $T(r), T(c)$ encode $m, m' \in [2_2^n]$. Let $b, b_1, b_2, b_3 \in \hat{\Gamma}$.*

1. *$\mathcal{P}_{red}$ accepts $T(\texttt{check}\ \ r\ c)$ from state $q_l^{\#}$ iff either $m' = 0$ or $m' = 2_2^n - 1$.*
2. *$\mathcal{P}_{red}$ accepts $T(\texttt{check}\ \ r\ c)$ from state $q_l^{\square}$ iff both $2 \leq m' < 2_2^n - 1$ and $m = 0$.*
3. *$\mathcal{P}_{red}$ accepts $T(\texttt{check}\ \ r\ c)$ from state $q_l^{(q_I,\square)}$ iff both $m = 0$ and $m' = 1$.*
4. *$\mathcal{P}_{red}$ accepts $T(\texttt{next}\ \ r\ c)$ from state $q^{(b_1,b_2,b_3)}$ iff (I) both $m > 0$ and $0 < m' < 2_2^n - 1$ and (II) $\mathcal{P}_{red}$ accepts $T\big(F\,(\texttt{dec}\ \ r)\ (\texttt{dec}\ \ c)\big)$ from state q^{b_1} and $T\big(F\,(\texttt{dec}\ \ r)\ c\big)$ from state $q^{b_2}\big)$ and $T\big(F\,(\texttt{dec}\ \ r)\ (\texttt{inc}\ \ c)\big)$ from state q^{b_3}.*
5. *$\mathcal{P}_{red}$ accepts $T(F\ r\ c)$ from state q^b iff $C(m, m') = b$.*

Finally, $\mathcal{P}_{red}$ accepts $T(\mathcal{G}_{red})$ iff $C(2_2^n - 1, 0) = (q_{acc}, \square)$.

The proof is in App. A.2.

This lemma then immediately yields the hardness result:

▶ **Theorem 14.** *It is 2-EXPTIME-hard to decide whether a given APT-TA pair accepts a given timed HORS.*

Proof. Take any DTM $\mathcal{M}$. By Prop. 10, it is 2-EXPTIME-hard to decide, given n, whether there is a successful 2_2^n-certificate for $\mathcal{M}$. By Lemma 13, this is equivalent to deciding whether $\mathcal{P}_{\text{red}}$, together with its trivial companion TA, accepts $T(\mathcal{G}_{\text{red}})$. The size of $\mathcal{P}_{\text{red}}$ is polynomial in $|\mathcal{M}|$, the size of the TA is polynomial in n, and the size of $\mathcal{G}_{\text{red}}$ is linear in n, since the bound `max` can be encoded in binary. Hence, this is a polynomial-time reduction to the timed-HORS model-checking problem which, in turn, must be 2-EXPTIME-hard. ◀

6 Conclusion

We have introduced the concept of timed APT and timed HORS, to be model-checked in conjunction with a TA. To our knowledge, this constitutes the first formalization of dense real time for the verification of trees generated by HORS. We have shown that real time adds at most one exponent to the model-checking problem, and that this is strict at order 1. This is in line with findings for settings where not the structure to be verified is higher-order, but the property a structure is verified against. For example, the model-checking problem for Recursive CTL is complete for exponential time [8], and adding real time to obtain Timed Recursive CTL makes the problem complete for two-fold exponential time [10].

While this paper introduces the concept of adding real time to recursion schemes, the expressiveness of the framework needs to be explored. For example, we could allow the APT to reset some clocks of its companion TA, which would make it easier to model certain problems. We would also like to invert the semantics of the TA in the sense that $\forall$ controls the flow of time. This would allow us to model safety properties of the form "no matter how long it takes to perform an action, correctness is guaranteed". However, the concept of timed automata is somewhat alien to adverse time flow, hence we expect such a change to be non-trivial.

We conjecture that the hardness result can be obtained for all k, not just for $k = 1$. This would be done by replacing the trees that encode large numbers by functions that consume such a tree and return such a tree, and functions that consume such functions and return trees encoding a number, etc. Such an approach has been used in similar settings [18, 11]. We also conjecture the existence of a restricted version of timed HORS and the APT-TA pairs used in the model-checking problem, such that the problem becomes characteristic not of time, but of space. This is suggested by similar fragments both for the timed setting [9] and for the model-checking problem for untimed HORS [7].

References

1 Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993. doi:10.1006/inco.1993.1024.

2 Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990. doi:10.1007/BFB0032042.

3 Rajeev Alur and P. Madhusudan. Decision problems for timed automata: A survey. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2004. doi:10.1007/978-3-540-30080-9_1.

4 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

5 Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. C-shore: a collapsible approach to higher-order verification. In Greg Morrisett and Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 13–24. ACM, 2013. doi:10.1145/2500365.2500589.

6 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICS.CSL.2013.129.

7 Florian Bruse. Space-efficient model-checking of higher-order recursion schemes. manuscript submitted.

8 Florian Bruse and Martin Lange. Temporal logic with recursion. *Inf. Comput.*, 281:104804, 2021. doi:10.1016/J.IC.2021.104804.

9 Florian Bruse and Martin Lange. The tail-recursive fragment of timed recursive CTL. *Inf. Comput.*, 294:105084, 2023. doi:10.1016/J.IC.2023.105084.

10 Florian Bruse and Martin Lange. Model checking timed recursive CTL. *Inf. Comput.*, 298:105168, 2024. doi:10.1016/J.IC.2024.105168.

11 Florian Bruse, Martin Lange, and Étienne Lozes. Space-efficient fragments of higher-order fixpoint logic. In Matthew Hague and Igor Potapov, editors, *Reachability Problems - 11th International Workshop, RP 2017, London, UK, September 7-9, 2017, Proceedings*, volume 10506 of *Lecture Notes in Computer Science*, pages 26–41. Springer, 2017. doi:10.1007/978-3-319-67089-8_3.

12 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

13 Werner Damm. The IO- and oi-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982. doi:10.1016/0304-3975(82)90009-3.

14 Gérard Huet. *Résolution d'Équations dans des Langages d'Order 1, 2., ω*. PhD thesis, Universite de Paris VII, 1976.

15 Marcin Jurdzinski. Small progress measures for solving parity games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000. doi:10.1007/3-540-46541-3_24.

16 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

17 Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20:1–20:62, 2013. doi:10.1145/2487241.2487246.

18 Naoki Kobayashi, Étienne Lozes, and Florian Bruse. On the relationship between higher-order recursion schemes and higher-order fixpoint logic. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 246–259. ACM, 2017. doi:10.1145/3009837.3009854.

19 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.29.

20 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.

A Omitted Proofs

A.1 Proofs for Section 4

▶ **Lemma 3.** *Let $\mathcal{A}$ be a TA, $\mathcal{P}$ a Ξ-APT, $\mathcal{G}$ a timed HORS, all over matching alphabets and clocks. Let m be greater than or equal to the index of $\mathcal{A}$, any interval on a tree constructor in $\mathcal{G}$, and any clock constraint in Ξ. Let v be a node in $T(\mathcal{G})$, q a state of $\mathcal{P}$, ℓ a location in $\mathcal{A}$ and $\eta \simeq_m \eta'$ two clock evaluations. Then $\exists$ wins the acceptance game from position $(v, q, (\ell, \eta), 0)$ iff she wins the game from position $(v, q, (\ell, \eta'), 0)$.*

Proof. We show the result using extra-clock semantics. Let $\mathcal{A}$, $\mathcal{P}$, $\mathcal{G}$ and m, η, η' be as in the lemma. We show the following, stronger result: $\exists$ wins the acceptance game from any position of the forms $(v, q, (\ell, \eta), 0)$ or $(v, \varphi, (\ell, \eta), 1)$ iff she wins from positions $(v, q, (\ell, \eta'), 0)$, resp. $(v, \varphi, (\ell, \eta'), 1)$. The proof is by induction over plays of the acceptance game. Since the claim of the lemma is fully symmetric, it suffices to show that a winning strategy for η yields one for η'. The invariant will be that $\exists$ keeps the game for η' in positions of the form $(v, q, (\ell, \eta'), 0)$ or $(v, q, (\ell, \eta'), 1)$ such that she wins the respective game from the position with η' replaced by some η with $\eta \simeq_m \eta'$. Clearly, the initial position for the claim of the lemma has this form.

The interesting case is that of a position of the form $(v, q, (\ell, \eta'), 0)$ with $l(v) = a_J$. By assumption, $\exists$ has a winning strategy from $((v, q, (\ell, \eta), 0)$. This means there is a trace $(\ell, \eta|_{\mathtt{z}}) = (\ell_1, \eta_1), \ldots, (\ell_n, \eta_n)$ where all the ℓ_i satisfy q and (ℓ_i, η_i) reaches (ℓ_{i+1}, η_{i+1}) either via a delay transition or a discrete transition, and $\eta_n(\mathtt{z}) \in J$. We produce a sequence $(\ell, \eta'|_{\mathtt{z}}) = (\ell_1, \eta'_1), \ldots, (\ell_n, \eta'_n)$ that satisfies the same conditions and, moreover, satisfies $\eta'_i \simeq_m \eta_i$ for all i.

It is easy to see that $\eta|_{\mathtt{z}} \simeq_m \eta'|_{\mathtt{z}}$. The rest of the sequence is generated by induction. If the next pair in the sequence is reached by a discrete transition for the η_i, then the same transition is available for η'_i since clock evaluations in the same region satisfy the same clock constraints (which appear as guards and location invariants here). If the next pair is reached via a delay transition using delay d, there is some d' such that delaying form η'_i reaches an equivalent region. This is due to the definition of $\simeq_m$: letting time flow in equivalent regions will pass through the same sequences or regions, since clocks reach integral values in the same order. Hence, such a sequence $(\ell_1, \eta'_1), \ldots, (\ell_n, \eta'_n)$ exists. Since $\eta'_n \simeq \eta_n$, also $\eta'_n(\mathtt{z}) \in J$. This finishes the case of positions of the form $(v, q, (\ell, \eta'), 0)$.

For the other cases, the invariant is easy to maintain since the TA part of the position does not change. This finishes the proof. ◀

▶ **Lemma 7.** *Let $\mathcal{G}, \mathcal{A}$ and $\mathcal{P}$ be a timed HORS, a TA and an APT over matching alphabets. $\mathcal{P}_\mathcal{A}$ accepts the tree generated by $\mathcal{G}^u$ iff the APT-TA pair accepts $T_\mathcal{G}$.*

Proof. The invariant for $\exists$ to maintain for either direction of the proof is to stay in a position $(v, (q, \ell, [\eta]))$ such that she wins from position $(v, \varphi, (\ell, \eta), 0)$ or $(v, q, (\ell, \eta), 1)$ in $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$, or to stay in a position of the latter form such that she wins from position $(v, (q', \ell, [\eta]))$ or $(v, (q, \ell, [\eta]))$ in the acceptance game $G(\mathcal{P}_\mathcal{A}, \mathcal{G}^u)$, where $q \in Q$, the state set of $\mathcal{P}$.

The interesting part is that of a position of the form $(v, (q, \ell, [\eta])$ in $G(\mathcal{P}_\mathcal{A})$. Assume that there is a corresponding position $(v, q, (\ell, \eta), 0)$ such that $\exists$ wins from there in $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$. Using extra-clock semantics, the game first transitions to $(v, q, (\ell, \eta|_\mathtt{z}), 0)$. Then, $\exists$ lets time flow, visiting only locations where q holds, up to some $(v, q, (\ell', \eta'), 0)$ with $\eta'(\mathtt{z}) \in J$ where $a_J = l(v)$. By Prop. 5, there is a path corresponding to the trace taken by $\exists$ in $G(\mathcal{P}_\mathcal{A}, \mathcal{G})$ passing through the respective regions, ending up in $(\ell', [\eta'])$. It is not hard to see that all regions on this path also must satisfy the proposition q. Hence, $\exists$ can move through the gadget defined by $\mathtt{start}^J$ by letting the initial transition for reset reset the clock $\mathtt{z}$, passing from $[\eta]$ to $[\eta|_\mathtt{z}]$, then choosing the leftmost branch at flow choosing successor pairs or regions and locations following the trace from $G(\mathcal{P}_\mathcal{A}, \mathcal{G}^u)$, and exiting via the right branch once she has replicated the trace. This yields a position $(v, (q, \ell', [\eta']))$ and by the definition of $\simeq_m$, also $\eta'(\mathtt{z}) \in J$, whence the check at check^J is passed.

The other transitions concern only the transition function of the original APT and do not manipulate time, whence it is not hard to verify the invariant. The converse direction follows a similar pattern, also invoking Prop. 5. This finishes the proof. ◀

A.2 Proofs for Section 5

▶ **Lemma 12.** *Let e be a closed term of type $\bullet$ such that $T(e)$ encodes some number $k \in [2^{2^n}]$. Then*

1. *$\mathcal{P}_{cnt}$ accepts $T(\mathtt{exists}\ e)$ from states q_i^{yes} with its clock set to some value $k \in [2^n]$ iff it accepts $T(e)$ from q_i^{yes} with the clock set to some integral k' with $k < k' \leq 2^n - 1$, and it accepts from states q_i^{no} iff it accepts $T(e)$ from q_i^{no} for no such integral clock value.*
2. *$\mathcal{P}_{cnt}$ accepts $T(\mathtt{all}\ e)$ from states q_i^{yes} with its clock set to some value $k \in [2^n]$ iff it accepts $T(e)$ from q_i^{no} and for all integral clock values k' with $k < k' \leq 2^n - 1$, and it accepts from states q_i^{no} iff it rejects $T(e)$ from q_i^{no} for some such integral clock value.*
3. *$T(\mathtt{inc}\ e)$ encodes the number $k + 1 \mod 2^{2^n}$ and $T(\mathtt{dec}\ e)$ encodes the number $k - 1 \mod 2^{2^n}$.*
4. *$\mathcal{P}_{cnt}$ accepts $T(\mathtt{isZero?}\ e)$ with the clock set to 0 from states q_i^{yes} iff $T(e)$ encodes 0, and from states q_i^{no} iff the tree generated by $T(e)$ encodes a number different from 0.*

Proof. The proof is by verification. We begin with the first item. The second item is shown similarly to the first item.

Let k be as in the lemma and assume that there is k' with $k < k' \leq 2^n - 1$ such that the automaton accepts $T(e)$ from q_i^{yes} . Let $n = k' - k$. We can construct a winning play for $\exists$ by choosing the pair $(q_1^{\text{yes}}, q^\top)$ for $n - 1$ times when reading the and_1. For each such transition, time passes for one unit, so $\mathtt{x}$ has value $k' - 1$ after doing this. Then let $\exists$ choose $(q^\top, q_1^{\text{yes}})$ at the next occurrence of $a_1^\vee$. Since $k' \leq 2^n - 1$, a winning play for the tree $\mathsf{and}\ \mathtt{max}\ x$ is easily constructed from the winning play for $T(e)$ from clock value k'. Conversely, assume that $\mathcal{P}_{\text{cnt}}$ accepts $T(\mathtt{exists}\ e)$ from q_i^{yes} from some clock value $k \in [2^n]$. There are two cases: either $\exists$ chooses $(q^\top, q_1^{\text{yes}})$ eventually, or she always chooses $(q_1^{\text{yes}}, q^\top)$. Note that, since the priority of q_1^{yes} is 1, this is a losing play. Hence, $\exists$ chooses the first option after n many choices of the second option, whence time has passed by $n + 1$ when reaching the tree

$T(\mathsf{and}\ \mathtt{max}\ x)$. Clearly, $k+n+1 \leq 2^n-1$ for otherwise the play is obviously not winning. Hence $k' = k+n+1$ is as desired. The statement for the state q_i^{no} is proved in a similar fashion.

The third item follows from the first and second items and the pattern of binary incrementation outlined above, the last item follows from the second item and Def. 11. ◀

▶ **Lemma 13.** *Let $C\colon 2_2^n \times 2_2^n \to \hat{\Gamma}$ be the unique certificate for n and $\mathcal{M}$. Let $r, c\colon \bullet$ be expressions such that $T(r), T(c)$ encode $m, m' \in [2_2^n]$. Let $b, b_1, b_2, b_3 \in \hat{\Gamma}$.*

1. *$\mathcal{P}_{red}$ accepts $T(\mathtt{check}\ \ r\ c)$ from state $q_l^{\#}$ iff either $m' = 0$ or $m' = 2_2^n - 1$.*
2. *$\mathcal{P}_{red}$ accepts $T(\mathtt{check}\ \ r\ c)$ from state $q_l^{\square}$ iff both $2 \leq m' < 2_2^n - 1$ and $m = 0$.*
3. *$\mathcal{P}_{red}$ accepts $T(\mathtt{check}\ \ r\ c)$ from state $q_l^{(q_I,\square)}$ iff both $m = 0$ and $m' = 1$.*
4. *$\mathcal{P}_{red}$ accepts $T(\mathtt{next}\ \ r\ c)$ from state $q^{(b_1,b_2,b_3)}$ iff (I) both $m > 0$ and $0 < m' < 2_2^n - 1$ and (II) $\mathcal{P}_{red}$ accepts $T\big(F\,(\mathtt{dec}\ \ r)\ (\mathtt{dec}\ \ c)\big)$ from state q^{b_1} and $T\big(F\,(\mathtt{dec}\ \ r)\ c\big)$ from state $q^{b_2}\big)$ and $T\big(F\,(\mathtt{dec}\ \ r)\ (\mathtt{inc}\ \ c)\big)$ from state q^{b_3}.*
5. *$\mathcal{P}_{red}$ accepts $T(F\ r\ c)$ from state q^b iff $C(m, m') = b$.*

Finally, $\mathcal{P}_{red}$ accepts $T(\mathcal{G}_{red})$ iff $C(2_2^n - 1, 0) = (q_{acc}, \square)$.

Proof. The first three items are shown directly. We show the first item, the other two are shown similarly. Note that the root $T(\mathtt{check}\ \ r\ c)$ is labeled by $\mathsf{or3}$, and that $\delta(q_l^{\#}, \mathsf{or3}) = (q_1^{\mathrm{yes}}, q^{\top}, q^{\top})$. Since every tree is accepted by $q^{\top}$, it remains to argue that the automaton accepts $T\big(\mathsf{and}(\mathtt{zero}\ \ (\mathtt{inc}\ \ r)\ \mathtt{zero}\ \ r)\big)$ from q_1^{yes} iff $m' = 0$ or $m' = 2_2^n - 1$. The former is equivalent to the automaton accepting $T(\mathtt{zero}\ \ (\mathtt{inc}\ r))$ and $T(\mathtt{zero}\ \ r)$ from q_1^{yes}. By Lemma 12, this is the case iff m', i.e., the number encoded by r, is either $2_2^n - 1$ or 0, which is what we wanted to show.

The proof for the last two items is by simultaneous induction over m'. Before we begin the induction consider the structures of the respective trees.

The tree root of $T(\mathtt{next}\ \ r\ c)$ is labeled by and, and both its left and right subtrees are labeled by $\mathsf{and3}$. We have $\delta(q^{(b_1,b_2,b_3)}, \_, \mathsf{and}) = (q_1^{\mathrm{yes}}, q^{(b_1,b_2,b_3)})$ and $\delta(q^{(b_1,b_2,b_3)}, \_, \mathsf{and3}) = (b_1, b_2, b_3)$, we obtain that $\mathcal{P}_{\mathrm{red}}$ accepts $T(\mathtt{next}\ \ r\ c)$ iff it accepts the following:

- $T(\mathsf{neg}\ (\mathtt{isZero?}\ r))$ from q_1^{yes},
- $T(\mathsf{neg}\ (\mathtt{isZero?}\ c))$ from q_1^{yes},
- $T(\mathtt{ltMax?}\ c)$ from q_1^{yes},
- $T(F\ (\mathtt{dec}\ r)\ (\mathtt{dec}\ c))$ from q^{b_1},
- $T(F\ (\mathtt{dec}\ r)\ c)$ from q^{b_2}, and
- $T(F\ (\mathtt{dec}\ r)\ (\mathtt{inc}\ c))$ from q^{b_3}.

By Lemma 12, the first three are equivalent to $m > 0$ and $0 < m' < 2_2^n - 1$, as in the claim.

Similar considerations yield that $\mathcal{P}_{\mathrm{red}}$ accepts the tree generated by $T(F\ r\ c)$ iff it accepts one of $T(\mathtt{next}\ \ r\ c)$ and $T(\mathtt{check}\ \ r\ c)$. Closer inspection of the requirements w.r.t. m and m yield that at most one of the latter can be the case (since acceptance of $T(\mathtt{next}\ \ r\ c)$ requires $m > 0$ and $0 < m' < 2_2^n - 1$, while acceptance of $T(\mathtt{check}\ \ r\ c)$ requires $m = 0$ or $m' = 0$ or $m' = 2_2^n - 1$).

Now let $m = 0$. The fourth item follows readily from the previous considerations, since both acceptance of $T(\mathtt{next}\ \ r\ c)$ and the premise of the item require $m > 0$. For the fifth item, the claim follows since only acceptance of $T(\mathtt{check}\ \ r\ c)$ is in question (since $m > 0$), and the first three items cover exactly the possibilities for $C(0, m')$, namely $\#$ (for $m' = 0$ and $m' = 2_2^n - 1$), $(q_I, \square)$ (for $m' = 1$) and $\square$ (for $1 < m' < 2_2^n - 1$).

Now let $m > 0$ and assume that the fourth and fifth item have been shown for $m - 1$. For the fourth item, the claim now follows easily from the itemized list above and the induction hypothesis applied to the last three items of the list. For the fifth item of the lemma, if $m' = 0$ or $m' = 2_2^n - 1$, apply the first item of the lemma, and if $0 < m' < 2_2^n - 1$, apply the fourth item of the lemma.

The last, unitemized claim then follows straightforwardly. ◀