History-deterministic Timed Automata

Thomas A. Henzinger
IST Vienna, Austria

Karoliina Lehtinen
CNRS, Aix-Marseille University, University of Toulon, LIS

Patrick Totzke
University of Liverpool, UK

Abstract

We explore the notion of history-determinism in the context of timed automata (TA). History-deterministic automata are those in which nondeterminism can be resolved on the fly, based on the run constructed thus far. History-determinism is a robust property that admits different game-based characterisations, and history-deterministic specifications allow for game-based verification without an expensive determinization step.

We show yet another characterisation of history-determinism in terms of fair simulation, at the general level of labelled transition systems: a system is history-deterministic precisely if and only if it fairly simulates all language smaller systems.

For timed automata over infinite timed words it is known that universality is undecidable for Büchi TA. We show that for history-deterministic TA with arbitrary parity acceptance, timed universality, inclusion, and synthesis all remain decidable and are ExpTime-complete.

For the subclass of TA with safety or reachability acceptance, we show that checking whether such an automaton is history-deterministic is decidable (in ExpTime), and history-deterministic TA with safety acceptance are effectively determinizable without introducing new states.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Timed Automata, History-determinism, Good-for-games, fair simulation, synthesis

1 Introduction

Automata offer paradigmatic formalisms both for specifying and for modelling discrete transition systems, *i.e.* for providing descriptive as well as executable definitions of formal languages. Given a finite or infinite word, an automaton specifies whether or not the word belongs to the defined language. Deterministic automata are executable, because the word can be processed left-to-right, with each transition of the automaton determined by the current input letter. Descriptive automata allow the powerful concept of nondeterminism, which yields more succinct or even more expressive specifications.

The notion of *history-determinism* lies between determinism and nondeterminism. History-deterministic automata are still executable, provided the execution engine is permitted to keep a record of all past inputs. Formally, a strategy r (*a.k.a.* "resolver") is a function from finite prefix runs to transitions that suggests for each input word w a specific run $r^*(w)$ of the automaton over w, namely, the run that results from having the function r determine, after each input letter, the next transition based on the prefix of the word processed so far. An automaton is *history-deterministic* if there exists a resolver r so that for every input word w, the automaton has an accepting run over w iff the specific run $r^*(w)$ is accepting.

The concept of history-determinism was first identified in [21], where it was noted that for solving graph games, it is not necessary to determinize history-deterministic specifications of ω-regular winning conditions. For this reason, history-deterministic automata were called "good-for-games". The term "history-determinism" was first used by [13]. The concept itself has since been referred to as both "history-determinism" and "good-for-gameness." Since [10]

recently showed that, in a general context of quantitative automata, the two notions do not always coincide (specifically: for certain quantitative winning conditions, history-determinism implies the "good-for-games" property of an automaton, but not vice versa), we follow their more nuanced terminology and use the term "history-determinism" to denote the existence of a resolver and "good-for-games" for automata that preserve the winner of games under composition, as required for solving games without determinization.

There is also a tight link between a variant of the Church synthesis problem, called *good-enough synthesis* [2], and deciding history-determinism. Church synthesis asks whether a system can guarantee that its interaction with an uncontrollable environment satisfies a specification language for all possible environment behaviours. This model assumes that the environment is hostile and will, if possible, sabotage the system's efforts. This pessimistic view can be counter-productive. In the canonical example of a coffee machine, if the users (the environment) do not fill in the water container, the machine will fail to produce coffee. Church synthesis would declare the problem unrealisable: the machine may not produce coffee for all environment behaviours. In the good-enough synthesis problem, on the other hand, such failures are acceptable, and we can still return an implementation that produces coffee (satisfies the specification) whenever the environment behaves in a way that allows the desired behaviour (fills in the water container). Deciding the good-enough synthesis problem for a deterministic automaton is polynomially equivalent to deciding whether a nondeterministic automaton of the same type is history-deterministic [16, 10, 18]. The decidability and complexity of checking history-determinism is therefore particularly interesting.

In this paper, we study, for the first time, history-determinism in the context of *timed* automata. In a timed word, letters alternate with time delays, which are nonnegative real numbers. The resolver gets to look not only at all past input letters, but also at all past time delays, to suggest the next transition. We consider timed automata over infinite timed words with standard ω-regular acceptance conditions [3]. For the results of this paper, it does not matter whether or not the sum of all time delays provided by an infinite input word is required to diverge.

Our results can be classified into two parts. The first part of our results applies to all timed automata, and sometimes more generally, to all labelled transition systems. In this part we are concerned with solving the quintessential verification problem for timed systems, namely *timed language inclusion*, in the special case of history-deterministic (*i.e.* executable) specifications. Since universality is undecidable for general timed automata, so is the timed language-inclusion problem for nondeterministic specifications [3]. This is the reason why much previous work in timed verification has focused on identifying determinizable subclasses of timed automata, such as event-clock automata [4], and on studying deterministic extensions of the timed-automaton model, such as deterministic two-way timed automata [5]. Determinizable specifications can be complemented, thus supporting the *complementation-based* approach to language inclusion: in order to check if every word accepted by the implementation A is also accepted by the specification B, first determinize and complement B, and then check the intersection with A for emptiness. We show that the history-determinism of specifications suffices for deciding timed language inclusion, which demonstrates that determinizability is not required. More precisely, we prove that if A is a timed automaton and B is a history-deterministic timed automaton, it can be decided in ExpTime if every timed word accepted by A is also accepted by B (Corollary 18).

In contrast to the traditional complementation-based approach to language inclusion, the history-deterministic approach is *game-based*. Like the complementation-based approach, the game-based approach is best formulated in the generic setting of labelled transition

systems with acceptance conditions, so-called *fair LTS*. The acceptance condition of a fair LTS declares a subset of the infinite runs of the LTS to be fair (a special case is *safety* acceptance, which declares all infinite runs to be fair). Given two fair LTS A and B, the language of A is included in the language of B if for every fair run of A there is a fair run of B over the same (infinite) word. A sufficient condition for the language inclusion between A and B is the existence of a fair simulation relation between the states of A and the states of B, or equivalently, the existence of a winning strategy for player p_B in the following 2-player *fair simulation game*: (i) every transition chosen by player p_A on the state-transition graph A can be matched by a transition chosen by player p_B on the state-transition graph B with the same label (letter or time delay), and (ii) if the infinite sequence of transitions chosen by p_A produce a fair run of A, then the matching transitions chosen by p_B produce a fair run of B [20]. Solving the fair simulation game is often simpler than checking language inclusion; it may be polynomial where language inclusion is not (*e.g.* in the case of finite safety or Büchi automata), or decidable where language inclusion is not (*e.g.* in the case of timed safety or Büchi automata [28]).

We show that for all fair LTS A and all history-deterministic fair LTS B, the condition that the language of A is included in the language of B is equivalent to the condition that A is fairly simulated by B. This observation reduces the language inclusion problem for history-deterministic specifications to the problem of solving a fair simulation game between implementation and specification. The solution of fair simulation games depends on the complexity of the acceptance conditions of A and B, but is often simpler than the complementation of B, and fair simulation games can be solvable even in the case of specifications that cannot be complemented. In the concluding Section 7, we conjecture the existence of such a timed language. The game-based approach to checking language inclusion, which requires history-determinism, is therefore more general, and often more efficient, than the traditional complementation-based approach to checking language inclusion, which usually requires full determinization. Indeed, history-determinism is exactly the condition that allows the game-based approach to language inclusion: for a given fair LTS B, if it is the case that B can fairly simulate all fair LTS A whose language is included in the language of B, then B must be history-deterministic (Theorem 4).

More generally, turn-based timed games for which the winning condition is defined by a history-deterministic timed automaton are no harder to solve than those with deterministic winning conditions: the winner of such a timed game can be determined on the product of the (timed) arena with the automaton specifying the winning condition. We conjecture that this is the case also for the concurrent timed games of [14] (cf. Section 7). Timed games have also been defined for the synthesis of timed systems from timed I/O specifications. Again, we show that the synthesis game of [15] can be solved not only for I/O specifications that are given by deterministic timed automata, but more generally, for those given by history-deterministic timed automata (Theorem 20).

The second part of our results investigates the problem of deciding history-determinism for timed automata and the determinizability of history-deterministic timed automata. In this part, we have only partial results, namely results for timed safety and reachability automata. Timed safety automata, in particular, constitute an important class of specifications, as many interesting timed and untimed properties can be specified by timed safety automata if time is required to diverge [19]. We prove that for timed safety automata and timed reachability automata, it can be decided in ExpTime if a given timed automaton is history-deterministic (Theorem 16). Checking history-determinism remains open for more general classes of timed automata, such as timed Büchi and coBüchi automata. We also show that every history-

deterministic timed safety automaton can be determinized, without increase in the state-space, but with an exponential increase in the number of transitions or length of guards (Theorem 9). While the question of determinizability is undecidable for nondeterministic timed reachability automata [17], it is open for history-deterministic timed reachability automata and for history-deterministic timed automata with more general acceptance conditions. Finally, we show that if a timed safety or reachability automaton is good-for-games (in the sense explained earlier), then the automaton must be history-deterministic (Theorem 23). This implication is open for more general classes of timed automata.

Related work. The notion of history-determinism was introduced independently, with slightly different definitions, by Henzinger and Piterman [21] for solving games without determinization, by Colcombet [13] for cost-functions, and by Kupferman, Safra, and Vardi [24] for recognising derived tree languages of word automata. Initially, history-determinism was mostly studied in the ω-regular setting, where these different definitions all coincide [9]. For some coBüchi-recognisable languages, history-deterministic automata can be exponentially more succinct than any equivalent deterministic automaton [23], and for Büchi and coBüchi automata, history-determinism is decidable in polynomial time [6, 23]. For transition-based history-deterministic automata, minimisation is PTime [1], while for state-based ones, it is NP-complete [27]. Recently, the notion has been extended to richer automata models, such as pushdown automata [25, 18] and quantitative automata [10, 11], where deterministic and nondeterministic models have different expressivity, and therefore, allowing a little bit of nondeterminism can, in addition to succinctness, also provide more expressivity.

Paper Structure. After defining preliminary notions we proceed to introduce history-determinism, and show a new, fair-simulation-base characterisation in Section 3. In Section 4 we demonstrate that history-deterministic TA with safety acceptance are determinizable and in Section 5 that one can decide whether a given safety or reachability TA is history-deterministic. Section 6 considers the language inclusion and synthesis problems and shows that history-determinism coincides with good-for-gameness for reachability and safety TA.

2 Preliminaries

Numbers, Words. Let $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ denote the nonnegative integers and reals, respectively. For $c \in \mathbb{R}_{\geq 0}$ we write $\lfloor c \rfloor$ for its integer and $\mathit{fract}(c) \stackrel{\text{def}}{=} c - \lfloor c \rfloor$ for its fractional part.

An alphabet Σ is a nonempty set of letters. Σ_ε denotes $\Sigma \cup \{\varepsilon\}$. Σ^* and Σ^ω denote the sets of finite and infinite words over Σ, respectively and $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ denotes their union. The empty word is denoted by ε, the length of a finite word v is denoted by $|v|$, and the n-th letter of a finite or infinite word is denoted by $w[n]$ (starting with $n = 0$).

Labelled Transition Systems, Languages, Fair Simulation. A *labelled transition system* (LTS) is a graph $S = (V, \Sigma, E)$ with set V of states and edges $E \subseteq V \times \Sigma \times V$, labelled by alphabet Σ. It is *deterministic* if for all $(s, a) \in V \times \Sigma$ there is at most one s' with $s \xrightarrow{a} s'$, and *complete* if for all $(s, a) \in V \times \Sigma$ there is at least one s' with $s \xrightarrow{a} s'$. We henceforth consider only complete LTSs. Together with an *acceptance condition* $\mathit{Acc} \subseteq E^\omega$ this can be used to define languages over Σ as usual: a word $w = l_0 l_1 \ldots \in \Sigma^\omega$ is accepted from s_0 if there is a path (also *run*) $\rho = s_0 \xrightarrow{l_1} s_1 \xrightarrow{l_2} s_2 \ldots$ that is accepting, i.e., in Acc. The *language* $L(s_0) \subseteq \Sigma^\omega$ of an initial state $s_0 \in V$ consists of all words for which there exists an accepting run from s_0. We will write $s \subseteq_L s'$ to denote language inclusion, meaning $L(s) \subseteq L(s')$. The acceptance condition Acc can be given by a parity condition but does not have to be. We consider in this paper especially reachability (does the run visit a state in a

given target set $T \subseteq V?$) and safety conditions (does the run always stay in a "safe" region $F \subseteq V?$). An LTS together with an accepting condition is referred to as *fair* LTS [20].

Fair simulations [20] are characterised by simulation games on (a pair of) fair LTSs in which Player 1 stepwise produces a path from s, and Player 2 stepwise produces an equally labelled path from s'. Player 2 wins if she produces an accepting run whenever Player 1 does. That is, s is fairly simulated by s' (write $s \preceq s'$) iff Player 2 has a strategy in the simulation game so that, whenever the run produced by Player 1 is accepting then so is the run produced by Player 2 in response. Fair simulation $s \preceq s'$ implies language inclusion $L(s) \subseteq L(s')$ but not vice versa.

Timed Alphabets, Words, and LTSs. For any alphabet Σ let Σ_T denote the timed alphabet $\{(a,t) | a \in \Sigma, t \in \mathbb{R}_{\geq} 0\}$. A timed word is a finite or infinite word $w \in (\Sigma_T)^\infty$ consisting of letters in Σ paired with distinct non-negative non-decreasing real-valued timestamps. We will also write $d_0 a_0 d_1 a_1 ...$ to denote a timed word $(a_i, t_i) \in \Sigma_T^\infty$ where $t_0 = d_0$ and $t_{i+1} = t_i + d_{i+1}$. Conversely, the duration and the timed word of any sequence in $(\Sigma \cup \mathbb{R})^\infty$ is given inductively as follows. For any $d \in \mathbb{R}_{\geq 0}$, $\tau \in \Sigma$, $\alpha \in (\Sigma \cup \mathbb{R})^*$, and $\beta \in (\Sigma \cup \mathbb{R})^\infty$ let $\mathit{duration}(\tau) \stackrel{\text{def}}{=} 0$; $\mathit{duration}(d) \stackrel{\text{def}}{=} d$; $\mathit{duration}(\alpha\beta) = \mathit{duration}(\alpha) + \mathit{duration}(\beta)$; $\mathit{tword}(\varepsilon) = \mathit{tword}(d) \stackrel{\text{def}}{=} \varepsilon$; $\mathit{tword}(\alpha d) \stackrel{\text{def}}{=} \mathit{tword}(\alpha)$; and $\mathit{tword}(\alpha\tau) \stackrel{\text{def}}{=} \mathit{tword}(\alpha)(\tau, \mathit{duration}(\alpha))$. An infinite timed word of finite duration is called a *zeno* word. Our results hold whether time must diverge (*i.e.*, zeno words are not considered) or not; we note whenever time divergence affects proofs.

A *timed* LTS is one with edge labels in $\Sigma \uplus \mathbb{R}_{\geq 0}$, so that edges labelled by $\mathbb{R}_{\geq 0}$ (modelling the passing of time) satisfy the following conditions for all $\alpha, \beta, \gamma \in V$ and $d, d' \in \mathbb{R}_{\geq 0}$.

1. (Zero-delay): $\alpha \xrightarrow{0} \alpha$,
2. (Determinism): If $\alpha \xrightarrow{d} \beta \wedge \alpha \xrightarrow{d} \gamma$ then $\beta = \gamma$,
3. (Additivity): $\alpha \xrightarrow{d} \beta \xrightarrow{d'} \gamma$ then $\alpha \xrightarrow{d+d'} \gamma$.

The timed language $L(s) \subseteq \Sigma_T^\omega$ of a state s consists of all the timed words read along accepting runs $L(s) \stackrel{\text{def}}{=} \mathit{tword}(L(s))$. We write $L(S)$ for the timed language of the initial state of the LTS S.

Timed Automata. Timed automata are finite-state automata equipped with finitely many real-valued variables called *clocks*, whose transitions are guarded by constraints on clocks. Constraints on clocks $C = \{x, y, \ldots\}$ are (in)equalities $x \triangleleft n$ where $x \in C$, $n \in \mathbb{N}$ and $\triangleleft \in \{\leq, <\}$. Let $\mathcal{B}(C)$ denote the set of Boolean combinations of clock constraints, called *guards*. A clock *valuation* $\nu \in \mathbb{R}^C$ assigns a value $\nu(x)$ to each clock $x \in C$. We write $\nu \models g$ if ν satisfies the guard g. A timed automaton (TA) $\mathcal{T} = (Q, \iota, C, \Delta, \Sigma, \mathit{Acc})$ is given by

- Q a finite set of states including an initial state ι;
- Σ an input alphabet;
- C a finite set of clocks;
- $\Delta \subseteq Q \times \mathcal{B}(C) \times \Sigma \times 2^C \times Q$ a set of transitions; each transition is associated with a guard, a letter, and a set of clocks to reset. A transition that reads letter $a \in \Sigma$ will be called an a-transition. We assume that for all $(s, \nu, a) \in Q \times \mathbb{R}^C_{\geq 0} \times \Sigma$ there is at least one transition $(s, g, a, r, s') \in \Delta$ so that ν satisfies g.
- $\mathit{Acc} \subseteq \Delta^\omega$ an acceptance condition.

Timed automata induce timed LTSs, and can thus be used to define timed languages, as follows. A *configuration* is a pair consisting of a control state and a clock valuation. These can evolve in two ways, as follows. For all configurations $(s, \nu) \in Q \times \mathbb{R}^C_{\geq 0}$,

- there is a *delay* step $(s, \nu) \xrightarrow{d} (s, \nu + d)$ for every $d \geq 0$, which increments all clocks by d.
- there is a *discrete* step $(s, \nu) \xrightarrow{\tau} (s', \nu')$ if $\tau = (s, g, a, r, s') \in \Delta$ is a transition so that ν satisfies g and $\nu' = \nu[r \to 0]$, that is, it maps r to 0 and agrees with ν on all other values.

Naturally, each delay d yields a unique successor configuration and $\nu \xrightarrow{d} \xrightarrow{d'} \nu' \iff \nu \xrightarrow{d+d'} \nu'$ for any two $d, d' \geq 0$ and valuations ν, ν'. So this indeed induces a timed LTS.

Discrete steps, however, are a source of nondeterminism: a configuration may have several a-successors induced by different transitions whose guards are satisfied. $\mathcal{T}$ is *deterministic* if its induced LTS is deterministic, which is the case iff for every state s, all transitions from s have mutually exclusive guards.

A path $\rho = (s_0, \nu_0) \xrightarrow{l_1} (s_1, \nu_1) \xrightarrow{l_2} (s_2, \nu_2) \dots$ is called *reduced* if it does not contain consecutive delay steps. It is a *run on* timed word $w \in (\Sigma_T)^\infty$ if $tword(l_1 l_2 \dots) = w$. The acceptance condition is lifted to the LTS as expected. Namely, a run is *accepting* if $\rho \in Acc$. This way, the *language* $L(s, \nu) \subseteq \Sigma_T^\omega$ of a configuration (s, ν) consists of all timed words for which there exists an accepting run from (s, ν). The language of $\mathcal{T}$ is $L(\mathcal{T}) \stackrel{\text{def}}{=} L((\iota, 0))$, the languages if the initial configuration with state ι and all clocks set to zero.

3 History-determinism

Informally, an automaton or LTS is history-deterministic if the non-determinism can be resolved on-the-fly, based only on the history of the word and run so far. We give two equivalent definitions, each being more convenient than the other for some technical developments.

▶ **Definition 1** (History-determinism). *A fair LTS $S = (V, \Sigma, E)$ is* history-deterministic *(from initial state $s_0 \in V$) if there is a* resolver $r : E^* \times \Sigma \to E$ *that maps every finite run and letter $a \in \Sigma$ to an a-labelled transition such that, for all words $w = a_0 a_1 \dots \in L(s_0)$ the run ρ defined inductively for $i > 0$ by $\rho_{i+1} \stackrel{def}{=} \rho_i r(\rho_i, a_{i+1})$, is an accepting run on w from s_0.*

Equivalently (from [9] for ω-regular automata), a resolver corresponds exactly to a winning strategy for Player 2 in the following *letter game*.

▶ **Definition 2** (Letter game). *The letter game on a fair LTS $S = (V, \Sigma, E)$ with initial state $s_0 \in V$ is played between Players 1 and 2. At turn i:*

- *Player 1 chooses a letter $a_i \in \Sigma$.*
- *Player 2 chooses an a_i labelled edge $\tau_i \in E$.*

A play is a pair (w, ρ) where $w = a_0 a_1 \dots$ is an infinite word and $\rho = \tau_0 \tau_1 \dots$ is a run on w. A play is winning for Player 2 if either $w \notin L(s_0)$ or ρ is an accepting run on w from s_0.

In these and other games we consider, strategies for both players are defined as usual, associating finite histories (runs) to valid player choices. Now winning strategies for Player 2 in the letter game exactly correspond to resolvers for S and vice-versa.

▶ **Proposition 3.** *Player 2 wins the letter game on S if and only if S is history-deterministic.*

While history-determinism is known to relate to fair simulation, in the sense that history-deterministic automata simulate deterministic ones for the same language [21], their relation has so far not been studied in more details. Below we show that history-determinacy can equivalently be characterised in terms of fair simulation.

▶ **Theorem 4.** *For every fair LTS S and initial state q the following are equivalent:*

1. *S is history-deterministic.*
2. *For all complete fair LTS S' with initial state q', $q' \subseteq_L q$ if and only if $q' \preceq q$.*

Proof (1) $\implies$ (2). Fair simulation $q \preceq q'$ trivially implies $q \subseteq_L q'$ by definition.

For the other implication, assume that $q \subseteq_L q'$. By assumption (1) there exists a resolver, i.e. a winning strategy in the letter game. Player 2 can win the fair simulation game

by ignoring her opponent's configuration and moving according to this resolver. By the completeness assumption on S', Player 1 can never propose a letter for which there is no successor in S'. So each player produces an infinite run on the same word w and the run produced by Player 2 is the same as that produced by the resolver in S'. If $w \in L(q)$ then it is in $L(q')$ and Player 2's run accepts. If $w \notin L(q)$ then Player 2 wins due to the fairness condition. In both cases she wins the fair simulation game and therefore $q \preceq q'$.

(2) $\implies$ **(1)** If condition (2) holds for all complete fair LTSs then q can fairly simulate the one consisting of a single state with self-loops for all transitions of S whose acceptance condition contains exactly all accepting runs from q. Then the strategy for Player 2 in the fair simulation game can be used as a strategy in the letter game. ◀

4 Expressivity

In this section we show that history-deterministic timed automata with safety acceptance are determinizable. To do so, we show (in Lemma 8) that these automata have simple resolvers, which only depend on the equivalence class of the current clock configuration with respect to the region abstraction. That is to say, the resolver only needs to know the integer part of clock values (up to the maximal value that appears in clock constraints) and the ordering of their fractional parts. We can then use such a simple resolver to determinize the automaton by adding guards that restrict transitions so that the automaton can only take one transition per region, as dictated by the resolver.

The following is the standard definition of regions (cf. [3], def. 4.3).

▶ **Definition 5** (Region abstraction). *Let $\mathcal{T} = (Q, \iota, C, \Delta, \Sigma, Acc)$ be a timed automaton and for any clock $x \in C$ let c_x denote the largest constant in any clock constraint involving x. Two valuations $\nu, \nu' \in \mathbb{R}^C_{\geq 0}$ are* (region) equivalent *(write $\nu \sim \nu'$) if all of the following hold.*

1. *For all $x \in C$ either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or both $\nu(x)$ and $\nu'(x)$ are greater than c_x.*
2. *For all $x, y \in C$ with $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, $fract(\nu(x)) \leq fract(\nu(y))$ iff $fract(\nu'(x)) \leq fract(\nu'(y))$.*
3. *For all $x \in C$ with $\nu(x) \leq c_x$, $fract(\nu(x)) = 0$ iff $fract(\nu'(x)) = 0$.*

Two configurations (q, ν) and (q', ν') are (region) equivalent, write $(q, \nu) \sim (q', \nu')$, if $q = q'$ and $\nu \sim \nu'$.

▶ **Definition 6** (Run-trees). *A* run-tree *on a timed word $u = (a_0, t_0)(a_1, t_1) \ldots$ from TA configuration (s_0, ν_0) is a tree where nodes are labelled by configurations, and edges by transitions such that*

1. *The labels along every branch form a run on u from (s_0, ν_0)*
2. *It is complete wrt. discrete steps: suppose the path leading towards some node is labelled by a run ρ which reads $tword(\rho) = (a_0, t_0) \ldots (a_i, t_i)$, ends in a configuration (s, ν), and has $duration(\rho) = t_{i+1}$. Then for every transition $\tau = (s, g, a_{i+1}, r, s') \in \Delta$ with $\nu \models g$ and so that $(s, \nu) \xrightarrow{\tau} (s', \nu')$, there is a τ-labelled edge to a new node labelled by (s', ν').*

A run-tree is reduced *if all its branches are. That is, there are no consecutive delay steps.*

Notice that for every initial configuration and timed word, there is a unique reduced run-tree, all of whose branches are runs on the word (since we have no deadlocks), and vice versa, all reduced runs on the word appear as branches on the run-tree.

We extend the region equivalence from configurations to run-trees in the natural fashion: two run-trees are equivalent if they are isomorphic and all corresponding configurations are equivalent. That is, they can differ only in fractional clock values and the duration of delays.

The following is our key technical lemma.

▶ **Lemma 7.** *Consider two region equivalent configurations* $(s, \nu) \sim (s', \nu')$.
For every timed word u *there is a timed word* u' *so that the reduced run-tree on* u *from* (s, ν) *is equivalent to the reduced run-tree on* u' *from* (s', ν').

Proof sketch. It suffices to show that for some (not necessarily reduced) run-tree on u from (s, ν) there exists some equivalent run-tree from (s', ν') as this implies the claim by collapsing all consecutive delay steps and thus producing the reduced tree on both sides.

We proceed by stepwise uncovering a suitable run-tree from (s, ν) for ever longer prefixes of u and constructing a corresponding equivalent run-tree from (s', ν'). The intermediate finite trees we build have the property that all branches have the same duration. In each round we extend all current leafs, in both trees, either by

1. all possible non-deterministic successors (for the letter prescribed by the word u), in case the duration of the branch is already equal to the next time-stamp in u, or
2. one successor configuration due to a delay, which must be *the same on all leafs*.

For the second case, the delays used to extend the two trees need not be the same because we only want to preserve region equivalence. Also, the delay chosen for the tree rooted in (s, ν) need not follow the timestamps in u but can be shorter, meaning the run-tree may not be reduced.. The difficulty lies in systematically choosing the delays to ensure that the two trees remain equivalent and secondly, that in the limit this procedure generates a run-tree on the whole word u from (s, ν). Together this implies the existence of a corresponding word u' and a run-tree from (s', ν').

To this end we propose a stronger invariant, namely that the relative orderings of the fractional values *in all leafs are the same on both sides*. The delays will be chosen in such a way as to always increase the maximal fractional clock value among all leafs to the next higher integer. Due to space constraints full details are deferred to Appendix A. ◀

We are now ready to show that history-deterministic TA with safety acceptance have simple resolvers based on the region abstraction.

▶ **Lemma 8.** *Every history-deterministic TA with safety acceptance has a resolver* r *that bases its decision only on the current letter and region. That is, for any letter* $a \in \Sigma$ *and any two finite runs* $(\iota, 0) \xrightarrow{\rho} (s, \nu)$ *and* $(\iota, 0) \xrightarrow{\rho'} (s', \nu')$ *consistent with* r *and so that* $(s, \nu) \sim (s', \nu')$, *it holds that* $r(\rho, a) = r(\rho', a)$.

Proof. Let r be a resolver for a history-deterministic safety TA $\mathcal{T}$.

We now build a resolver that only depends on the region of the current configuration. To do so, we choose a representative configuration within each region, which will determine the choice of the resolver for the whole region: For every region $R \in [Q \times \mathbb{R}^C_{\geq 0}]_\sim$, consider the configurations that are reached by at least one r-consistent run, and mark one of them m_R, if at least one exists, along with one r-consistent run ρ_R leading to the configuration m_R.

Let r' be the aspiring resolver that, when reading a letter a, considers the region R of the current configuration, and follows what r does when reading a after the marked r-consistent run ρ_R. We set $r'(\rho, a) \stackrel{\text{def}}{=} r(\rho_R, a)$ where R is the final region of the prefix-run ρ. Note that r' is well defined since it always follows transitions consistent with some r-consistent run and can therefore only visit marked regions.

We claim that r' is indeed a resolver. Towards a contradiction, assume that it is not a resolver, that is, there is some word $w \in L(\mathcal{T})$ for which r' builds a rejecting run. As $\mathcal{T}$ is a

safety automaton, we can consider the last configuration (s, ν) along this run from which the remaining suffix au of w can be accepted [1].

Suppose that ρ is the prefix of the run built by r' on w, which ends in (s, ν) and let $\tau = r'(\rho, a)$ be the a-transition chosen by r'. We know that τ leads from (s, ν) to some configuration (s', ν') from where u is not accepted. By definition of r', there must be a marked configuration $m_R \sim (s, \nu)$ reached by some run ρ_R from which r chooses the same a-transition τ. By Lemma 7 there must be a word au' so that the run-tree on au from (s, ν) is equivalent to that on au' from m_R. This means that $au' \in L(m_R)$ and, as r is a resolver, there must be an accepting run that begins with a step $(m_R) \xrightarrow{\tau} (m'_R)$. We derive that u also has an accepting run from (q, ν) that begins with τ, contradicting the assumption that (q, ν) is the last position on the run r' built on w so that its suffix can be accepted. Therefore, r' is indeed a resolver. ◀

We can now use the region-based solver to determinize history-deterministic safety TA.

▶ **Theorem 9.** *Every history-deterministic safety TA is equivalent to a deterministic TA.*

Proof. Consider a history-deterministic TA $\mathcal{T} = (Q, \iota, C, \Delta, \Sigma, Acc)$, with a region-based resolver (as in Lemma 8) r, and let R be the region graph of $\mathcal{T}$. Define $\mathcal{T}' = (Q, \iota, C, \Delta', \Sigma, Acc)$ where $(q, g \wedge z, a, X, q') \in \Delta'$ for z a guard defining a region of R, that is, a guard that is satisfied exactly by valuations in R, if $(q, g, a, X, q') \in \Delta$ is the transition chosen by r in the region defined by the guard z. In other words, $\mathcal{T}'$ is $\mathcal{T}$ with duplicated transitions guarded so that a transition can only be taken from a region from which r chooses that transition. Observe that $\mathcal{T}'$ is deterministic: the guards describing regions are mutually exclusive, therefore the guards of any two transitions from the same state over the same letter have mutually exclusive guards.

As runs of $\mathcal{T}'$ corresponds to a run of $\mathcal{T}$ with added guards, $L(\mathcal{T}') \subseteq L(\mathcal{T})$. Conversely, if $w \in L(\mathcal{T})$, then its accepting run consistent with r is also an accepting run in $\mathcal{T}'$, since each transition along this run, being chosen by r, is taken at a configuration that satisfies the additional guards in $\mathcal{T}'$. We can therefore conclude that $L(\mathcal{T}) = L(\mathcal{T}')$. ◀

While this determinization procedure preserves the state-space of the automaton, it multiplies the number of transitions (or the size of guards) by the size of the region abstraction. Then, while history-deterministic safety TA are no more expressive than deterministic ones, they could potentially be exponentially more succinct, when counting transitions and guards.

5 Deciding History-determinism

Recall the letter game characterisation of history-determinism: Player 1 plays timed letters and Player 2 responds with transitions. Player 2 wins if either the word is not in the language of the automaton, or her run is accepting. As TA are not closed under complement, it isn't clear how to solve this game. Bagnol and Kuperberg [7] introduced *token games*, which are easier to solve, but which coincide with the letter game for some types of automata, in particular for Büchi [7], coBüchi [8] and some quantitative automata [11].

[1] The fact that a rejecting run produced by a non-resolver must ultimately reach a configuration that cannot accept the remaining word also holds for TAs over finite words. However, this is *not* the case for reachability acceptance, which is why we only state the claim for safety here. Still, we conjecture that history-deterministic TA with reachability acceptance admit region-based resolvers.

In the k-token game, in addition to providing letters, Player 1 also builds k runs, of which at least one should be accepting. The fewer runs Player 1 is allowed to use, the more information he gives Player 2 about the word he will play. We show that the 1 and 2-token games characterize history-determinism for fair LTSs with safety and reachability acceptance.

▶ **Definition 10** (k-token game [6]). *Given a fair LTS $S = (V, \Sigma, E)$ with initial state $s_0 \in V$ and an integer $k > 0$, the game $G_k(S)$ proceeds in rounds. At each round i:*

- *Player 1 plays a letter $a_i \in \Sigma$*
- *Player 2 plays a transition τ_i in E*
- *Player 1 plays transitions $\tau_{1,i}, \tau_{2,i} \dots \tau_{k,i}$ in S*

This way, Player 1 chooses an infinite word $w = a_0 a_1 \dots$ and exactly k runs $\rho_i = \tau_{i,0}\tau_{i,1}\tau_{i,2}\dots$ for $1 \leq i \leq k$, and Player 2 chooses a run $\rho = \tau_0\tau_1 \dots$. The play is winning for Player 1 if some ρ_j is an accepting run over $t_0 a_0 \dots$ from s_0 but ρ is not. Else it is winning for Player 2.

We write $G_k(\mathcal{T})$ to mean the k-token game on the LTS induced by $\mathcal{T}$.

▶ Remark 11. $G_k(S)$ and the letter game are determined for any k and fair LTS S for any Borel-definable acceptance condition [26]. In particular, the letter game is determined for both safety and reachability TA. Indeed, the winning condition for Player 2 is a disjunction of the complement of $L(\mathcal{B})$ and of the acceptance condition of $\mathcal{B}$. Then, as long as $L(\mathcal{B})$ is Borel, by the closure of Borel sets under complementation and disjunction, the letter-game is Borel, and therefore determined, following Martin's Theorem [26]. If time is not required to diverge, then reachability timed languages and safety timed languages are clearly Borel. Since words in which time diverges are also Borel (they can be seen as the countable intersection of words where time reaches each unit time), this remains the case when we require divergence.

The next lemma was first stated for finite [6], then for quantitative automata [11]. The same proof works for all (generally infinite) fair LTSs, and is given again in Appendix B.

▶ **Lemma 12.** *Given an fair LTS S, if Player 2 wins $G_2(S)$ then she wins $G_k(S)$ for all k.*

$G_1(S)$ was shown to characterise history-determinism for a number of quantitative automata in [11]. In Appendix B we show, using similar proof techniques, that this is also the case for all safety LTSs. The key observation is that for Player 2 to win the letter game, it suffices that she avoids mistakes. We then show that a winning strategy for her in $G_1(S)$ can be used to build such a strategy.

▶ **Lemma 13.** *Given a fair LTS S with a safety acceptance condition, Player 2 wins $G_1(S)$ if and only if S is history-deterministic.*

This argument does not work for reachability TA: it is no longer enough for Player 2 to avoid bad moves to win; she needs to also guarantee that she will actually reach a final state. Here, we characterise history-determinism with the 2-token game. However, our proof requires finite branching in Player 2's choices, so we can not state it for LTSs in general.

▶ **Lemma 14.** *Given a finitely branching fair LTS S with a reachability acceptance condition, Player 2 wins $G_2(S)$ if and only if S is history-deterministic.*

Proof. If Player 2 wins in the letter game, she wins in $G_2(S)$ by ignoring Player 1's tokens.

Else, since the letter game is determined (Remark 11), Player 1 wins in the letter game on S with a strategy σ. All plays that agree with σ must eventually play a good prefix, that is, a prefix of a timed word of which either all continuations are in $L(S)$ if time is not required to diverge, or all non-zeno continuations are in $L(S)$ if time is required to diverge. At each turn Player 2 has only a finite number of enabled transitions to choose from, because

S is finitely branching. Therefore the strategy-tree for σ is finitely branching and by König's lemma, there is a bound k such that any play that agrees with σ has played a good prefix after k steps.

We now argue that Player 1 wins in $G_{k'}(S)$ for a large enough k'. Let k' be larger than the number of distinct run prefixes of length k on any word of length k played by σ (that is, at most b^k where b is the branching degree of S). Then, in $G'_k(S)$, Player 1 wins by using the following strategy: he plays the letters according to σ and Player 2's moves and moves his k' tokens along all possible run prefixes for the first k moves, and then chooses transitions arbitrarily. Since after k steps σ guarantees that he has played a good prefix, at least one of his runs built in this manner is accepting.

This strategy is winning: indeed, if Player 2 could beat it with some strategy σ', then she could use σ' in the letter game to beat σ, a contradiction. From Lemma 12, and the determinacy of $G_k(S)$, Player 1 therefore wins $G_2(S)$ whenever he wins the letter game. ◀

We now consider the problem of deciding whether a given safety or reachability TA is history-deterministic. We use the observation that the k-token games played on LTSs induced by TA can be expressed as a timed parity game from [12] played on the $(k+1)$-fold product.

▶ **Lemma 15.** *For all k and timed safety or reachability automata $\mathcal{T}$, the game $G_k(\mathcal{T})$ is solvable in* ExpTime.

Proof. $G_k(\mathcal{T})$ is a timed game on an arena consisting of the configuration space of the product of $k+1$ copies of $\mathcal{T}$. The winning condition consists of a boolean combination of safety or reachability conditions. Such games can be solved as timed parity games as defined in [12] in time exponential in the number of clocks c and in k [12, Theorem 3]. Note that [12] uses concurrent timed parity games, of which turn-based ones are a special case. ◀

▶ **Theorem 16.** *Given a safety or reachability TA, deciding whether it is history-deterministic is decidable in* ExpTime.

Proof. From Lemma 13 and Lemma 14, deciding the history-determinism of a safety or reachability TA $\mathcal{T}$ reduces to solving $G_1(\mathcal{T})$ or $G_2(\mathcal{T})$ respectively, both of which can be done in ExpTime, from Lemma 15. ◀

As explained in the introduction, this also solves the good-enough synthesis problem of deterministic safety and reachability TA.

6 Synthesis, Games and Composition

In this section we consider several games played on (LTSs of) timed automata and how they can be used to decide classical verification problems. We focus on turn-based games, although our techniques can be generalised to concurrent ones. We first look at language inclusion, then synthesis, and finally we consider good-for-games timed automata, that is, automata that preserve the winner when composed with a game and show that good-for-gameness and history-determinism coincide for both reachability and safety timed automata.

6.1 Language Inclusion and Fair Simulation games

The connection between history-determinism and fair simulation, established in Theorem 4, allows to transfer decidability results to history-deterministic TA. Let's first recall that simulation checking is decidable for timed automata using a region construction [28]. This

paper precedes the notion of fair simulation (restricting Player 1 to fair runs) and is thus only applicable for safety conditions. However, the result holds for more general parity acceptance (for which each state is assigned an integer priority and where a run is accepted if the highest priority it sees infinitely often is even).

▶ **Lemma 17.** *Fair simulation is decidable and* ExpTime*-complete for parity timed automata.*

Proof. It suffices to observe that the simulation game can be presented as a timed parity game, as studied in [12], played on the product of two copies of the automaton. These can be solved in ExpTime. A matching lower bound holds even for safety or reachability acceptance (see Lemma 24 in Appendix C for details).

◀

▶ **Corollary 18.** *Timed language inclusion is decidable and* ExpTime*-complete for history-deterministic TA. More precisely, given a TA S with initial state q and a history-deterministic TA S' with initial state q', checking if $q \subseteq_L q'$ holds is* ExpTime*-complete.*

Proof. As $\mathcal{B}$ is history-deterministic and by Theorem 4, we have $q \subseteq_L q'$ if, and only if, $q \preceq q'$. The result follows from Lemma 17. ◀

6.2 Synthesis Games

We show that as is the case in the regular [21], pushdown [25], cost function [13], and quantitative [10] settings, synthesis games with winning conditions given by history-deterministic TA are no harder to solve than those with for winning condition given by deterministic TA.

▶ **Definition 19** (Timed synthesis game)**.** *Given a timed language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega_T$, the synthesis game for L proceeds as follows. At turn i:*

- *Player I plays a delay d_i and a letter $a_i \in \Sigma_I$*
- *Player II plays a letter $b_i \in \Sigma_O$.*

Player II wins if $d_0\binom{a_0}{b_0}d_1\binom{a_1}{b_1}... \in L$ or if time does not progress. If Player II has a winning strategy in the synthesis game, we say that L is realisable.

▶ **Theorem 20.** *Given a history-deterministic timed parity automaton $\mathcal{T}$, the synthesis game for $L(\mathcal{T})$ is decidable and* ExpTime*-complete.*

The proof (in Appendix C) follows a similar reduction to one in [25], in which the nondeterminism of the automaton is moved into Player 2's output alphabet, forcing her to simultaneously build a word in the winning condition and an accepting run witnessing this. Since accepting runs are recognised by deterministic automata, this reduces the problem to the synthesis problem for deterministic timed automata. The lower bound follows from the ExpTime-completeness of synthesis for deterministic TA [15].

The ExpTime decidability of universality for history-deterministic TA follows both from the decidability of language inclusion in the previous section and from the decidability of synthesis: the universality of $\mathcal{T}$ reduces to deciding the winner of the synthesis game over $\{\binom{w}{w} \mid w \in L(\mathcal{T})\}$, recognised by a history-deterministic TA if $\mathcal{T}$ is history-deterministic.

6.3 Composition with games

Implicitly, at the heart of these reductions is the notion of composition: the composition of the game to solve with a history-deterministic automaton for the winning condition yields an equivalent game with a simpler winning condition. We say that an automaton is

good-for-games if this composition operation preserves the winner of the game for all games. While history-determinism always implies good-for-gameness, the converse is not necessarily true. While the classes of history-deterministic and good-for-games automata coincide for ω-regular automata [9], this is not the case for quantitative automata [10], which can be good-for-games without being history-deterministic. We argue that for reachability and safety timed automata, good-for-gameness and history-determinism coincide.

▶ **Definition 21** (Timed Games). *A timed game (roughly following [15]), consists of an arena $\mathcal{G} = (Q, \iota, C, \Delta, \Sigma, L)$ and is similar to a TA except that Q, which need not be finite, is partitioned into $Q = Q_1 \uplus Q_2$, that is, positions Q_1 belonging to Player 1 and positions Q_2 belonging to Player 2, and L is a timed language, not an acceptance condition. Furthermore, an a-transition produces the letter a, rather than reads it. Configurations are defined as for TA and we assume every configuration to have at least one successor-configuration.*

A timed game proceeds in the configuration space of $\mathcal{G}$ with Player 1 at each turn i advancing time with a delay $d_i \in \mathbb{R}$. Then, from the resulting configuration c_i, the owner of the state of c_i chooses a transition in Δ enabled in c_i, leading to a transition c_{i+1} producing a letter a_i. An infinite play is winning for Player 2 if the word $d_0a_0d_1a_1\ldots$ produced is in L.

▶ **Definition 22** (Composition). *Intuitively, the composition of a game $\mathcal{G}$ and an automaton $\mathcal{T}$ consists of a game in which the two players play on $\mathcal{G}$ while Player 2 must also build, letter by letter, a run of $\mathcal{T}$ on the outcome of the game in $\mathcal{G}$. More formally, given a TA $\mathcal{T}$ and a game $\mathcal{G}$ with winning condition $L(\mathcal{T})$, the composition $\mathcal{T} \circ \mathcal{G}$ consists of a game played on the product of the configuration spaces of $\mathcal{G}$ and $\mathcal{T}$, starting from the initial state of both, in which, at each turn i, from a configuration (c_i, c'_i), Player 1 plays a time delay $d_i \in \mathbb{R}$, the owner of the current $\mathcal{G}$-state chooses a move in the configuration space of $\mathcal{G}$ to a successor-configuration c_{i+1}, producing a letter a_i, and then Player 2 chooses a transition over (d_i, a_i) enabled at the current $\mathcal{T}$-configuration c'_i, leading to a successor-configuration c'_{i+1}. The game then proceeds from (c_{i+1}, c'_{i+1}).*

Player 2 wins infinite plays if the run built in $\mathcal{T}$ is accepting, and loses if it is rejecting or if she cannot move in the $\mathcal{G}$-component.

Observe that if Player 1 wins in $\mathcal{G}$, then he also wins in $\mathcal{T} \circ \mathcal{G}$ with a strategy that produces a word not in $L(\mathcal{T})$ in $\mathcal{G}$, as then Player 2 can not produce an accepting run in $\mathcal{T}$.

[10, Lemma 7] shows that for (quantitative) automata for which the letter-game is determined, (threshold) history-determinism coincides with good-for-gameness. The lemma is stated for quantitative automata, where thresholds are relevant; in the Boolean setting, it simply states that the determinacy of the letter game implies the equivalence of history-determinism and good-for-gameness. In our timed setting, a similar argument, combined with the determinacy of the letter game for safety and reachability TA, gives us the following.

▶ **Theorem 23.** *Let $\mathcal{T}$ be a safety or reachability TA. The following are equivalent:*

1. *$\mathcal{T}$ is history-deterministic.*
2. *For all timed games $\mathcal{G}$ with winning condition $L(\mathcal{T})$, whenever Player 2 wins $\mathcal{G}$, she also wins $\mathcal{T} \circ \mathcal{G}$.*

Proof. (1) $\Longrightarrow$ (2) If $\mathcal{T}$ is history-deterministic, the resolver can be used as a strategy in the $\mathcal{T}$ component of $\mathcal{T} \circ \mathcal{G}$. When combined with a winning strategy in $\mathcal{G}$ that guarantees that the $\mathcal{G}$-component produces a word in $L(\mathcal{T})$, the resolver guarantees that the $\mathcal{T}$-component produces an accepting run, thus giving the victory to Player 2.

(2) $\Longrightarrow$ (1) Towards a contradiction, assume $\mathcal{T}$ is not history-deterministic, that is, by determinacy of the letter game from Remark 11, that Player 1 has a winning strategy σ in

the letter game. Now consider the game $\mathcal{G}_\sigma$, without clocks or guards, in which positions, all belonging to Player 1, consist of the prefixes of timed words played by σ, with moves $w \xrightarrow{(t,a)} w(t,a)$. As σ is winning for Player 1, all maximal paths in $\mathcal{G}_\sigma$ are labelled by a timed word in $L(\mathcal{T})$, so $\mathcal{G}_\sigma$ is winning for Player 2.

We now argue that Player 1 wins $\mathcal{T} \circ \mathcal{G}_\sigma$ by interpreting Player 2's moves in the $\mathcal{T}$ component as her moves in the letter game, and choosing moves in $\mathcal{G}$ mimicking the letter dictated by σ. Then, if Player 2 could win against this strategy in $\mathcal{T} \circ \mathcal{G}_\sigma$, she could also win against σ in the letter game by interpreting Player 1's choices of letters as moves in $\mathcal{G}$, and responding with the same transition as she plays in the $\mathcal{T}$ component of $\mathcal{T} \circ \mathcal{G}_\sigma$. Such a strategy is a valid strategy in the letter game on $\mathcal{T}$, and while it might not be winning in general, it is winning against σ, contradicting that σ is a winning strategy for Player 1. ◀

This proof fails for acceptance conditions beyond safety and reachability, as it isn't clear whether timed Büchi and coBüchi automata define Borel sets. If this was the case then history-deterministic timed automata would be exactly those that preserve winners in composition with games, as is the case in the ω-regular setting.

7 Conclusion

We introduced history-determinism for timed automata and showed that it suffices for solving important problems that previously required full determinism, in particular, timed language inclusion, universality and synthesis. We showed that for the important classes of timed safety and timed reachability automata, history-determinism can be checked (and therefore good-enough synthesis of deterministic reachability and safety automata can be solved) and every history-deterministic timed safety automaton can be determinized.

We conjecture that determinization does not hold for history-deterministic timed coBüchi automata. Consider the timed coBüchi language "there is a real time t such that for every nonnegative integer i, there is a letter a at time $t + i$." This timed language is recognised by a history-deterministic coBüchi automaton in which a nondeterministic transition guesses a "witness time" t after which a occurs at every unit interval, and which allows for an unbounded number of failed guesses (using the coBüchi condition). To see that this automaton is history-deterministic, let the resolver repeatedly and deterministically pick the time with the most previous occurences of a at unit-interval distances. If a timed input word is in the language, then this resolver will eventually choose a correct witness time and produce an accepting run.

We conjecture that the complement of this language cannot be defined by a (nondeterministic) timed automaton. Informally, a timed automaton would require an unbounded number of clocks to check that "for all occurrences of a there is a nonnegative integer distance i such that a is not followed by another a after i time units." If so, this timed language would separate the classes of deterministic and history-deterministic timed languages.

Let us conclude with another conjecture. We showed that history-deterministic timed automata are "good" for solving turn-based timed games, where in each turn of the game, one of the two players chooses a time delay or an action. A more general, concurrent setting for timed games is presented in [14]. In the concurrent version both players simultaneously choose permissible pairs of time delays and actions, and the player who has picked the shorter time delay gets to move. While concurrent games may not be determined, we conjecture that these concurrent timed games can again be solved by composing the (timed) arena with the (timed) winning condition, as long as the winning condition is history-deterministic.

References

1 Bader Abu Radi and Orna Kupferman. Minimizing gfg transition-based automata. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

2 Shaull Almagor and Orna Kupferman. Good-enough synthesis. In *CAV*, volume 12225 of *Lecture Notes in Computer Science*, pages 541–563. Springer, 2020.

3 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994. URL: http://www.sciencedirect.com/science/article/pii/0304397594900108, doi:10.1016/0304-3975(94)90010-8.

4 Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999. URL: https://doi.org/10.1016/S0304-3975(97)00173-4.

5 Rajeev Alur and Thomas A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *33rd Annual Symposium on Foundations of Computer Science*, pages 177–186. IEEE Computer Society, 1992. URL: https://doi.org/10.1109/SFCS.1992.267774.

6 Marc Bagnol and Denis Kuperberg. Büchi Good-for-Games Automata Are Efficiently Recognizable. In Sumit Ganguly and Paritosh Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, volume 122 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9915, doi:10.4230/LIPIcs.FSTTCS.2018.16.

7 Marc Bagnol and Denis Kuperberg. Büchi Good-for-Games Automata Are Efficiently Recognizable. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, volume 122 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9915, doi:10.4230/LIPIcs.FSTTCS.2018.16.

8 Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michal Skrzypczak. On succinctness and recognisability of alternating good-for-games automata. *CoRR*, abs/2002.07278, 2020. URL: https://arxiv.org/abs/2002.07278, arXiv:2002.07278.

9 Udi Boker and Karoliina Lehtinen. Good for games automata: From nondeterminism to alternation. In *Proceedings of CONCUR*, volume 140 of *LIPIcs*, pages 19:1–19:16, 2019.

10 Udi Boker and Karoliina Lehtinen. History Determinism vs. Good for Gameness in Quantitative Automata. In Mikołaj Bojańczy and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*, volume 213 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2021/15549, doi:10.4230/LIPIcs.FSTTCS.2021.38.

11 Udi Boker and Karoliina Lehtinen. Token games and history-deterministic quantitative automata. In Patricia Bouyer and Lutz Schröder, editors, *Foundations of Software Science and Computation Structures*, pages 120–139, Cham, 2022. Springer International Publishing. URL: https://doi.org/10.1007/978-3-030-99253-8_7.

12 Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Timed parity games: Complexity and robustness. In Franck Cassez and Claude Jard, editors, *Formal Modeling and Analysis of Timed Systems*, pages 124–140, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

13 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *International Colloquium on Automata, Languages, and Programming*, pages 139–150, 2009.

14 Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In *CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003. URL: https://doi.org/10.1007/978-3-540-45187-7_9.

15 Deepak D'souza and P. Madhusudan. Timed control synthesis for external specifications. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 571–582. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45841-7_47.

16 Emmanuel Filiot, Christof Löding, and Sarah Winter. Synthesis from weighted specifications with partial domains over finite words. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2020.

17 Olivier Finkel. Undecidable problems about timed automata. In *FORMATS*, volume 4202 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2006. URL: https://doi.org/10.1007/11867340_14.

18 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A Bit of Nondeterminism Makes Pushdown Automata Expressive and Succinct. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14493, doi:10.4230/LIPIcs.MFCS.2021.53.

19 Thomas A. Henzinger, Peter W. Kopke, and Howard Wong-Toi. The expressive power of clocks. In *ICALP*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer, 1995. URL: https://doi.org/10.1007/3-540-60084-1_93.

20 Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. Fair simulation. In *CONCUR '97: Concurrency Theory*, pages 273–287. Springer Berlin Heidelberg, 1997.

21 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In Zoltán Ésik, editor, *Computer Science Logic*, pages 395–410, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

22 Marcin Jurdzinski, Francois Laroussinie, and Jeremy Sproston. Model Checking Probabilistic Timed Automata with One or Two Clocks. *Logical Methods in Computer Science*, Volume 4, Issue 3, September 2008. URL: https://lmcs.episciences.org/988, doi:10.2168/LMCS-4(3:12)2008.

23 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *International Colloquium on Automata, Languages, and Programming.*, pages 299–310, 2015.

24 Orna Kupferman, Shmuel Safra, and Moshe Y Vardi. Relating word and tree automata. *Ann. Pure Appl. Logic*, 138(1-3):126–146, 2006. Conference version in 1996.

25 Karoliina Lehtinen and Martin Zimmermann. Good-for-games ω-pushdown automata. *Logical Methods in Computer Science*, 18, 2022.

26 Donald A Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

27 Sven Schewe. Minimising good-for-games automata is np-complete. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2020.

28 Serdar Tasiran, Rajeev Alur, Robert P. Kurshan, and Robert K. Brayton. Verifying abstractions of timed systems. In *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 546–562. Springer, 1996. URL: https://doi.org/10.1007/3-540-61604-7_75.

A Expressivity

▶ **Lemma 7.** *Consider two region equivalent configurations* $(s,\nu) \sim (s',\nu')$.

For every timed word u *there is a timed word* u' *so that the reduced run-tree on* u *from* (s,ν) *is equivalent to the reduced run-tree on* u' *from* (s',ν').

Proof. It suffices to show that for some (not necessarily reduced) run-tree on u from (s,ν) there exists some equivalent run-tree from (s',ν') as this implies the claim by collapsing all consecutive delay steps and thus producing the reduced tree on both sides.

We proceed by stepwise uncovering the run-tree from (s,ν) for ever longer prefixes of u and constructing a corresponding equivalent run-tree from (s',ν'). The intermediate finite trees we build have the property that all branches have the same duration. In each round we extend all current leafs, in both trees, either by

1. all possible non-deterministic successors (for the letter prescribed by the word u), in case the duration of the branch is already equal to the next time-stamp in u, or
2. one successor configuration due to a delay, which must be *the same on all leafs.*

For the second case, the delays used to extend the two trees need not be the same because we only want to preserve region equivalence. Also, the delay chosen for the tree rooted in (s,ν) need not follow the timestamps in u but can be shorter, meaning the run-tree may not be reduced.. The difficulty lies in systematically choosing the delays to ensure that the two trees remain equivalent and secondly, that in the limit this procedure generates a run-tree on the whole word u from (s,ν). Together this implies the existence of a corresponding word u' and a run-tree from (s',ν').

Invariant. To this end we propose a stronger invariant, namely that the relative orderings of the fractional values *in all leafs are the same on both sides.* To be precise, let's reinterpret a clock valuation as a function $\nu : C \times \mathbb{N} \to \{\bot\} \cup [0,1)$, that assigns to every clock and possible integral value either a fractional value between 0 and 1, or $\bot$ (indicating that the given clock does not have the given integral value). This way for every clock x there is exactly one $n \in \mathbb{N}$ with $\nu(x,n) \neq \bot$ and the image $\nu(C \times \mathbb{N})$ has at most $|C|+1$ different elements. For any ordered set $F = \{\bot < f_1 < f_2 < \cdots < f_l\} \supseteq \nu(C \times \mathbb{N})$ of fractional values, we can thus represent ν as a function $\hat{\nu} : C \times \mathbb{N} \to \{\bot, 1, \ldots l\}$ that, instead of exact fractional clock values only yields their index in F (and maps $\bot \mapsto \bot$).

Consider some run-tree with leafs $(q_1,\nu_1)(q_2,\nu_2)\cdots(q_l\nu_l)$ with combined fractional values $F = \bigcup_{i=1}^{l} \nu_i(C \times \mathbb{N})$, and an equivalent run-tree with leafs $(q'_1,\nu'_1)(q'_2,\nu'_2)\cdots(q'_l\nu'_l)$ with combined fractional values $F' = \bigcup_{i=1}^{l} \nu'_i(C \times \mathbb{N})$. The two trees are *aligned* if for all $1 \leq i \leq l$, $\hat{\nu}_i = \hat{\nu}'_i$. Notice that this still allows the two trees to differ on their exact fractional values but now they must agree on the relative order of all contained clocks on leafs, and in particular which ones are maximal and therefore the closest to the next larger integer. We will always select a delay of $1 - \max\{F\}$ and $1 - \max\{F'\}$, respectively, in step 2 above.

To show the claim we produce the required run-trees starting in $(s,\nu) \sim (s'\nu')$. These are in particular two aligned run-trees on the empty word.

Assume two aligned trees as above, where leafs have fractional values $F = \{\bot < f_1 < f_2 < \cdots < f_m\}$ and $F' = \{f'_0 < f'_1 < \cdots < f'_m\}$, respectively, and assume that the tree rooted in (s,ν) reads a strict prefix $(a_0,t_0),\ldots(a_i,t_i)$ of u.

Case 1: the duration of all branches in the first tree equals t_{i+1}, the timestamp of the next symbol in u. Then we extend each leaf in both trees by all possible a_{i+1}-successors. This will produce two aligned trees because each leaf configuration in one must be region equivalent to the corresponding configuration in the other, and therefore satisfies the same

guards, enabling the same a_{i+1}-transitions leading to equivalent successors. Note also that all branches in each tree still have the same duration, as no delay step was taken.

Case 2: the duration of all branches in the first tree is strictly less than t_{i+1}. Then we extend all leafs in the tree from (s, ν) by a delay of duration $d = 1 - f_m$ and all leafs in the other tree by a delay of duration $d' = 1 - f'_m$. Naturally, this produces exactly one successor for each former leaf. The sets of new fractional values on leafs are $\bigcup_{i=1}^{m}(\mu + d)(C \times \mathbb{N}) = \{\bot < 0 < f_1 + d < \cdots < f_{m-1} + d\}$ and for any former leaf (q, μ) extended by a delay $(q, \mu) \xrightarrow{d} (q, \mu + d)$, we have

$$\hat{\mu}(x, n-1) = m \iff \widehat{(\mu + d)}(x, n) = 0 \tag{1}$$

and

$$\hat{\mu}(x, n) = i < m \iff \widehat{(\mu + d)}(x, n) = i + 1 \leq m \tag{2}$$

Analogous equivalences hold for the corresponding step $(q, \mu') \xrightarrow{d'} (q, \mu' + d')$ on the other tree. Notice that the two cases above are exhaustive as again, for all $x \in C$ there is exactly one $n \in \mathbb{N}$ with $\mu(x, n) \neq \bot$. We aim to show that $\widehat{(\mu + d)} = \widehat{(\mu' + d')}$. Consider any $x \in C$ and $n \in \mathbb{N}$. We have that

$$\begin{aligned} \widehat{(\mu + d)}(x, n) = m &\overset{(1)}{\iff} \hat{\mu}(x, n+1) = 0 \\ &\overset{(IH)}{\iff} \hat{\mu'}(x, n+1) = 0 \\ &\overset{(1)}{\iff} \widehat{(\mu' + d')}(x, n) = m \end{aligned}$$

and

$$\begin{aligned} \widehat{(\mu + d)}(x, n) = i < m &\overset{(2)}{\iff} \hat{\mu}(x, n) = i + 1 \\ &\overset{(IH)}{\iff} \hat{\mu'}(x, n) = i + 1 \\ &\overset{(2)}{\iff} \widehat{(\mu' + d')}(x, n) = i < m \end{aligned}$$

It follows that $\widehat{(\mu + d)} = \widehat{(\mu' + d')}$ which means that the two trees are again aligned, as required.

To see why this procedure produces a run-tree on u (and an equivalent run-tree on some word u'), observe that there can be at most $|F| + 1$ many consecutive delay extensions according to step 2) before all integral clock values are strictly increased. ◀

B Deciding History-determinism

▶ **Lemma 12.** *Given an fair LTS S, if Player 2 wins $G_2(S)$ then she wins $G_k(S)$ for all k.*

This is the generalisation of [6, Thm 14] (on ω-regular automata) to fair LTSs. The proof is similar to [6], without requiring positional strategies, and identical to that of [11, Theorem 4] (on quantitative automata), without the quantitative aspects. If Player 2 wins $G_2(S)$ then she obviously wins $G_1(S)$, using her G_2 strategy with respect to two copies of Player 1's single token in G_1. We therefore consider below $k > 2$.

Let σ_2 be a winning strategy for Player 2 in $G_2(S)$. We inductively show that Player 2 has a winning strategy σ_i in $G_i(S)$ for each finite i. To do so, we assume a winning strategy σ_{i-1} in $G_{i-1}(S)$. The strategy σ_i maintains some additional (not necessarily finite) memory

that maintains the position of one virtual token in S, a position in the (not necessarily finite) memory structure of σ_{i-1}, and a position in the (not necessarily finite) memory structure of σ_2. The virtual token is initially at the initial state of S. Then, the strategy σ_i then plays as follows: at each turn, after Player 1 has moved his i tokens and played a letter (or, at the first turn, just played a letter), it first updates the σ_{i-1} memory structure, by ignoring the last of Player 1's tokens, and, treating the position of the virtual token as Player 2's token in $G_{i-1}(S)$, it updates the position of the virtual token according to the strategy σ_{i-1}; it then updates the σ_2 memory structure by treating Player 1's last token and the virtual token as Player 1's 2 tokens in $G_2(S)$, and finally outputs the transition to be played according to σ_2.

We now argue that this strategy is indeed winning in $G_i(S)$. Since σ_{i-1} is a winning strategy in $G_{i-1}(S)$, the virtual token traces an accepting run if any of the runs built by the first $i-1$ tokens of Player 1 is accepting. Since σ_2 is also winning, the run built by Player 2's token is accepting if either the run built by the virtual token or by Player 1's last token is accepting. Hence, Player 2's is accepting whenever one of Player 1's runs is accepting, making this a winning strategy in $G_i(S)$.

▶ **Lemma 13.** *Given a fair LTS S with a safety acceptance condition, Player 2 wins $G_1(S)$ if and only if S is history-deterministic.*

Proof. If S is history-deterministic then Player 2 wins $G_1(S)$ by using the resolver to choose her transitions. This guarantees that for all words in $L(S)$ played by Player 1, her run is accepting, which makes her victorious regardless of Player 1's run.

For the converse, if Player 2 wins $G_1(S)$, consider the following family of *copycat strategies* for Player 1: at first, Player 1 plays σ and chooses the same transitions as Player 2; if, eventually, Player 2 chooses a transition τ from a configuration c that is not language-maximal, that is, moves to a configuration c' that does no accept some word w that is accepted by some other configuration c'' reachable by some other transition τ' from c, we call such a move non-cautious, and Player 1 stops copying Player 2 and instead chooses τ'. From there, Player 1 wins by playing w and an accepting run on w from c''. Since Player 2 wins $G_1(S)$, her winning strategy σ does not play any non-cautious moves against copycat strategies.

Then, she can use σ in the letter-game, by playing as σ would play in $G_1(S)$ if Player 1 copies her transitions. This guarantees that she never makes a non-cautious move, and, in particular, never moves out of the safe region of the automaton unless the prefix played by Player 1 has no continuations in $L(S)$. This is a winning strategy in the letter-game, so S is history-deterministic. ◀

C Synthesis, Games and Composition

▶ **Theorem 20.** *Given a history-deterministic timed parity automaton $\mathcal{T}$, the synthesis game for $L(\mathcal{T})$ is decidable and* ExpTime*-complete.*

Proof. For the upper bound, we reduce the problem to solving synthesis games for deterministic timed parity automata, which is in ExpTime [15].

Let $\mathcal{T} = (S, \iota, C, \Delta, \Sigma, Acc)$ be a timed automaton. Let $\mathcal{T}'$ be the deterministic timed automaton $(S, \iota, C, \Delta', \Sigma \times \Delta, Acc)$ where:

$$\Delta' = \{(s, g, (\sigma, (s, g, \sigma, c, s')), c, s') | (s, g, \sigma, c, s') \in \Delta\}$$

In other words, $\mathcal{T}'$ is a deterministic automaton with the state space of $\mathcal{T}$, over the alphabet $\Sigma \times \Delta$, where the transition in the input letter dictates the transition in the

automaton. The language of $\mathcal{T}'$ is the set of words (w, ρ) such that there is an accepting run of $\mathcal{T}$ over w along the transitions of ρ.

We now claim that given a history-deterministic automaton $\mathcal{T}$ with resolver r, Player II wins the synthesis game on $\mathcal{T}$ if and only if she wins it on $\mathcal{T}'$. First assume that Player II wins the synthesis game for $\mathcal{T}$ with a strategy s. Then, to win the synthesis game for $\mathcal{T}'$, at each turn i, after Player I plays d_i and a_i, she needs to make two choices: she must choose both a response letter b_i and a transition in $\mathcal{T}$ over (a_i, b_i). Given Player I's move and the (first component of the) word built so far, she can use the strategy s to choose the response letter b_i; this guarantees that the first component of the play is a word accepted by $\mathcal{T}$. To choose the transition of $\mathcal{T}$, she can use the resolver r: given the run ρ built from the delays (including d_i) and transitions played so far, she plays $r(\rho, (a_i, b_i))$. Since r is a resolver, this strategy guarantees that the resulting run is accepting, and hence that she wins the synthesis game on $\mathcal{T}'$.

On the other hand, if Player I wins the synthesis game on $\mathcal{T}$, he has a strategy s which guarantees a play $w \in (\Sigma_i \times \Sigma_O)^T$ that is not in the language of $\mathcal{T}$. He can use the same strategy in the synthesis game of $\mathcal{T}'$ to guarantee a play (w, ρ) such that w is not in the language of $\mathcal{T}$, and by extension (w, ρ) is not in the language of $\mathcal{T}'$, as there are no accepting runs over w in $\mathcal{T}$.

The lower bound follows from the ExpTime-completeness of synthesis for deterministic TA [15]. ◀

Below we demonstrate that fair simulation checking for TA is ExpTime-hard even for very simple acceptance conditions.

▶ **Lemma 24.** *Checking fair simulation between TA is* ExpTime*-hard already for reachability or safety acceptance, or over finite words.*

Proof. This can be shown by reduction from *countdown games* [22], which are two-player games (Q, T, k) given by a finite set Q of control states, a finite set $T \subseteq (Q \times \mathbb{N}_{>0} \times Q)$ of transitions, labelled by positive integers, and a target number $k \in \mathbb{N}$. All numbers are given in binary encoding. The game is played in rounds, each of which starts in a pair (p, n) where $p \in Q$ and $n \leq k$, as follows. First Player 1 picks a number $l \leq k - n$, so that at least one $(p, l, p') \in T$ exists; Then Player 2 picks one such transition and the next round starts in $(p', n + l)$. Player 1 wins iff she can reach a configuration (q, k) for some state q.

Determining the winner in a countdown game is ExpTime-complete [22] and can easily encoded as a simulation game between two TAs $\mathcal{A}$ and $\mathcal{B}$ as follows. Let $\mathcal{A}$ be the TA with no clocks and unrestricted (guards are *True*) self-loops for the two letters a and e; The idea is that Player 1 proposes l by waiting that long and then makes a discrete a-labelled move. Then Player 2, currently in some state p can update his configuration to mimic that of the countdown game, and punish (by going to a winning sink) if Player 1 cheated or the game should end. To implement this, $\mathcal{B}$ has two clocks: one to store n – the total time that passed – and one to store the current l, which is reset in each round.

Suppose Player 1 waits for l units of time and then proposes a. Player 2, currently in some state p will have

- a and e-labelled transitions to a winning state with a guard that verifies that there is no transition (p, l, p').
- a-labelled transitions to a state p', with a guard that verifies that a some $(p, l, p') \in T$ exists, and which resets clock x_2.
- a, and e-labelled transitions to a winning state guarded by $x_1 > k$. This enables Player 2 to win if the global time has exceeded the target k.

The only way that Player 1 can win is by following a winning strategy in the countdown game and by playing the letter e once $\mathcal{B}$ is in a configuration (q, k). Player 2 will not be able to respond. ◀