

History-Determinism vs Fair Simulation

Udi Boker   


Reichman University, Herzliya, Israel

Thomas A. Henzinger   

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Karoliina Lehtinen   

CNRS, LIS, Aix-Marseille Univ., France

Aditya Prakash   

University of Warwick, Coventry, UK

Abstract

An automaton \mathcal{A} is history-deterministic if its nondeterminism can be resolved on the fly, only using the prefix of the word read so far. This mild form of nondeterminism has attracted particular attention for its applications in synthesis problems. An automaton \mathcal{A} is guidable with respect to a class C of automata if it can fairly simulate every automaton in C , whose language is contained in that of \mathcal{A} . In other words, guidable automata are those for which inclusion and simulation coincide, making them particularly interesting for model-checking.

We study the connection between these two notions, and specifically the question of when they coincide. For classes of automata on which they do, deciding guidability, an otherwise challenging decision problem, reduces to deciding history-determinism, a problem that is starting to be well-understood for many classes.

We provide a selection of sufficient criteria for a class of automata to guarantee the coincidence of the notions, and use them to show that the notions coincide for the most common automata classes, among which are ω -regular automata and many infinite-state automata with safety and reachability acceptance conditions, including vector addition systems with states, one-counter nets, pushdown-, Parikh-, and timed-automata.

We also demonstrate that history-determinism and guidability do not always coincide, for example, for the classes of timed automata with a fixed number of clocks.

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

Keywords and phrases History-Determinism

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.12

Related Version *Full Version*: <https://arxiv.org/abs/2407.08620> [3]

Funding *Udi Boker*: Israel Science Foundation grant 2410/22

Thomas A. Henzinger: ERC-2020-AdG 101020093 (VAMOS)

Karoliina Lehtinen: ANR QUASY 23-CE48-0008-01

Aditya Prakash: Chancellors' International Scholarship from the University of Warwick and Centre for Discrete Mathematics and Its Applications (DIMAP)

1 Introduction

Language inclusion between automata is a key problem in verification: given an automaton representing a program and another representing a specification, language inclusion of the former in the latter captures precisely whether all executions of the program satisfy the specification. Unfortunately, in the presence of nondeterminism, inclusion is algorithmically hard. For instance, for regular automata it is PSPACE-hard on both finite and infinite words.



© Udi Boker, Thomas A. Henzinger, Karoliina Lehtinen, and Aditya Prakash;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 12; pp. 12:1–12:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

12:2 History-Determinism vs Fair Simulation

Fair simulation is a more syntactic approximation of inclusion, defined by the simulation game [12]. In this game, one player, in the role of the spoiler, builds, transition by transition, a run in one of the automata, say \mathcal{A} , while the other, in the role of the duplicator, chooses at each turn a matching transition in the other automaton, say \mathcal{B} . The second player's task is to build a run that is accepting if the first player's run is accepting. If the duplicator has a winning strategy, then \mathcal{B} is said to simulate \mathcal{A} , which, in particular, implies that \mathcal{A} 's language is included in \mathcal{B} 's language. Simulation, due to its local and syntactic nature, is generally easier to check than inclusion: for instance it is in PTIME for nondeterministic Büchi automata. As a result, automata for which language inclusion and simulation coincide are particularly well-suited for model-checking. We call such automata *guidable*, after a similar notion used previously as an alternative to determinism for tree automata [9]. Despite their clear usefulness for model checking, guidable automata have so far been mostly used as a tool, but not studied much in their own right, with the notable exception of [17].

Guidability is not easy to decide: it is contingent on an automaton simulating a potentially infinite number of language-included automata. We would like to have, whenever possible, a characterisation that is more amenable to algorithmic detection.

Deterministic automata are of course always guidable, and so are *history-deterministic* automata. These are mildly nondeterministic automata, in which nondeterministic choices are permitted, but can only depend on the word read so far, rather than the future of the word. These automata have received a fair bit of attention recently due, in particular, to their applications in synthesis problems [6]. In general, they offer an interesting compromise between the power of nondeterministic automata and the better algorithmic properties of deterministic ones. In particular, they can simulate all equivalent, or language-contained, automata as they only need the history to resolve nondeterministic choice in the best possible way – in other words, they are guidable. In fact, at first it might appear that history-determinism and guidability should coincide; indeed, this is the case if we consider guidability with respect to all labelled transition systems [13, Theorem 4]. However, there are also classes of automata for which this is not the case.

Guidability and history-determinism coinciding on a class C of automata is equivalent to the description “for every automaton $\mathcal{A} \in C$ that is not history-deterministic, there is some $\mathcal{A}' \in C$ that is language-included in \mathcal{A} but that \mathcal{A} does not simulate” (D). Then it is easy enough to hand-pick automata to build classes where guidability and history-determinism do not coincide (for example, a class of inclusion-incomparable automata that are not all history-deterministic). However, as we will see, there are also more natural classes of automata, such as timed automata with a bounded number of clocks, for which guidability and history-determinism differ.

The characterisation (D) is too abstract to be much use for analysing the usual automata classes we are interested in. We therefore prove that several more concrete criteria (Theorem 1) imply that guidability and history-determinism coincide, and use each of these in a comprehensive analysis of standard automata classes. Roughly, each of the criteria describes some sufficient closure properties which guarantee the existence of \mathcal{A}' from description (D). If some automaton can simulate another automaton that is sufficiently difficult to simulate (for example a deterministic one, since they simulate all equivalent automaton), then it must be history-deterministic; as a result, a class of automata having sufficient closure properties (such as closure under determinisation), implies that guidability and history-determinism coincide. The challenge is to identify, for a variety of different classes of automata C , an automaton that is sufficiently difficult to simulate, while remaining in C .

In order to discuss our sufficient criteria in more detail, we need to start with a couple of key notions, the first of which is the *1-token game* and the second is the *1-token ghost*.

History-determinism of an automaton is tricky and expensive to decide directly [11]; As an alternative, Bagnol and Kuperberg used k -token games as potential characterisations for history-determinism [2]. Roughly, they resemble a (fair) simulation-like game, played on a single automaton, where one player, Eve (in the role of Duplicator), must build transition-by-transition a run on a word dictated letter-by-letter by Adam, who, after each of Eve's choices, also builds k runs transition-by-transition. Eve then wins if her run is accepting whenever Adam's run is accepting. Bagnol and Kuperberg showed that for Büchi automata, the 2-token game indeed characterises history-determinism, which means that deciding history-determinism for Büchi automata is in PTIME [2]. Since then, the 1- or 2-token games have been shown to characterise history-determinism for various automata classes, including coBüchi [4], DSum, LimInf, and LimSup automata [5]. These games contrast with the *letter game*, a game which always characterises history-determinism, but which is often challenging to solve directly [11].

In this work, we make heavy use of token-games, this time to understand the connection between history-determinism and guidability. In particular, we extend token games to be played over two automata (Definition 3), separating between Eve's and Adam's automata, and define, given an automaton \mathcal{A} , that an automaton \mathcal{A}' is a *1-token ghost* of \mathcal{A} if \mathcal{A}' is language-equivalent to \mathcal{A} and Eve wins the 1-token game between \mathcal{A}' and \mathcal{A} . For some classes of automata such a ghost is easy to construct, while for other classes it might not exist due to lacking closure properties.

With these notions, we can now state our criteria.

► **Theorem 1.** *The notions of history-determinism and guidability coincide for a class C of labelled transitions systems (LTSs) if at least one of the following holds:*

1. *Determinisation. C is closed under history-determinism, i.e., for each nondeterministic LTS in C , there is a language-equivalent history-deterministic (or deterministic) LTS in C as well. (Lemma 4)*
2. *1-token ghost. 1-token games characterise history-determinism in C , and C is closed under 1-token ghost. (Lemma 8)*
3. *Strategy ghost. For every $\mathcal{A} \in C$ that is not history-deterministic, there is a deterministic LTS \mathcal{B} over the alphabet of transitions of \mathcal{A} , such that \mathcal{B} recognises the plays of a winning strategy of Adam in the letter game on \mathcal{A} , and \mathcal{B} , projected onto the alphabet of \mathcal{A} , has a 1-token ghost in C . (Lemma 9)*

We prove the criteria of Theorem 1 in Section 4, and use them in Section 5 to show that the notions of history-determinism and guidability coincide for numerous classes of automata, listed in Table 1. We also provide counter-examples of classes for which history-determinism and guidability do not coincide, as listed in Table 1, and elaborated on in Section 6. We restrict our analysis to automata over infinite words, which are better behaved in this context, and discuss how to adapt our techniques for finite words in Section 7.

A practical corollary of our result is that guidability is decidable, with the complexity of deciding history-determinism, for ω -regular automata (EXPTIME for parity automata, PTIME for Büchi and coBüchi), safety and reachability timed automata (EXPTIME) and visibly pushdown automata (EXPTIME). For details on the complexity of these procedures, we refer the reader to a recent survey [6].

12:4 History-Determinism vs Fair Simulation

■ **Table 1** History-determinism vs guidability.

Automata Class	HD = Guidability by
ω -regular	<i>Determinisation or Strategy ghost</i>
Fixed-index parity (e.g., Büchi), Weak	<i>Strategy ghost</i> Corollary 10
Linear	<i>Strategy ghost</i> Theorem 19. (Not ghost-closed Theorem 18)
Safety & Reachability pushdown automata, VASS, timed, one-counter automata, one-counter net, Parikh	<i>1-token ghost</i> Theorem 17 and Corollary 15
VPA with any ω -regular acceptance condition	<i>Strategy ghost</i> Theorem 16
Classes for which HD \neq guidability:	
– Büchi automata with a bounded number of states. Theorem 20	
– Timed automata with a bounded number of clocks. Theorem 21	
Notable classes for which we leave the question HD \neq guidability open:	
– PDA/OCA/OCN/Timed automata with general ω -regular acceptance	

2 Preliminaries

We use \mathbb{N} and \mathbb{N}^+ to denote the set of non-negative and positive integers respectively. We use $[i..j]$ to denote the set $\{i, \dots, j\}$ of integers, and $[i]$ for the set $[1..i]$. An alphabet Σ is a non-empty set of letters. A finite or infinite word is a finite or infinite sequence of letters from Σ respectively. We let ε be the empty word, and Σ_ε the set $\Sigma \cup \{\varepsilon\}$. The set of all finite (resp. infinite) words is denoted by Σ^* (resp. Σ^ω). A *language* is a set of words.

2.1 Labelled transition systems

A *labelled transition system* (LTS) $\mathcal{A} = (\Sigma, Q, \iota, \Delta, \alpha)$ consists of a potentially infinite alphabet Σ , a potentially infinite state-space Q , an initial state $\iota \in Q$, a labelled transition relation $\Delta \subseteq Q \times \Sigma_\varepsilon \times Q$, and a set of accepting runs α , where a run ρ is a (finite or infinite) sequence of transitions starting in ι and following Δ . We may write $q \xrightarrow{\sigma} q'$ instead of $(q, \sigma, q') \in \Delta$ for $\sigma \in \Sigma_\varepsilon$. Given a finite run $\rho = q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_k} q_k$ on the word $v = \sigma_0 \cdot \sigma_1 \cdot \sigma_2 \cdot \dots \cdot \sigma_k \in \Sigma_\varepsilon^{k+1}$, we write $q_0 \xrightarrow{v, \rho} q_k$ to denote that ρ is a transition sequence on the word v that starts at q_0 and ends at q_k . An LTS is *deterministic* if for every state q and letter $\sigma \in \Sigma$, there is at most one transition $q \xrightarrow{\sigma} q'$ from q on the σ , and there are no transitions on ε .

A word $w \in \Sigma^\omega$ is accepted by an LTS \mathcal{A} if there is an accepting run of \mathcal{A} on w . The language $L(\mathcal{A})$ of \mathcal{A} is the set of words that it accepts. An LTS \mathcal{A} is contained in an LTS \mathcal{B} , denoted by $\mathcal{A} \leq \mathcal{B}$, if $L(\mathcal{A}) \subseteq L(\mathcal{B})$, while \mathcal{A} and \mathcal{B} are equivalent, denoted by $\mathcal{A} \equiv \mathcal{B}$ if $L(\mathcal{A}) = L(\mathcal{B})$.

A *transducer* is like an LTS without acceptance condition and ε -transitions but with an output labelling: it is given by a tuple $(\Sigma_I, \Sigma_O, Q, \iota, \Delta, \gamma)$, where Σ_I and Σ_O are the input and output alphabets, respectively, $\Delta \subseteq Q \times \Sigma_I \times Q$ is the transition relation, and $\gamma : \Delta \rightarrow \Sigma_O$ is the output function. A *strategy* in general is a deterministic transducer. It is finite memory if the transducer has a finite state space.

2.2 Automata

We briefly recall the automata types considered here, which we assume to operate on infinite words unless stated otherwise, and leave the formal definitions to the appendix.

LTSs are represented concisely by various automata. An automaton \mathcal{A} induces an LTS \mathcal{B} , whose states are the configurations of \mathcal{A} , and whose runs are the same as \mathcal{A} 's runs. If \mathcal{A} 's states and configurations are the same, as is the case with ω -regular automata which we define below, then \mathcal{B} is identical to \mathcal{A} , but with the acceptance condition given by a set of runs (as opposed to an ω -regular conditions). If the configurations of \mathcal{A} contain additional data, as is the case for example with pushdown automata, then \mathcal{B} and \mathcal{A} have different states. Notice that \mathcal{A} is deterministic iff \mathcal{B} is. The acceptance condition of \mathcal{A} induces the acceptance condition on \mathcal{B} .

In a parity condition, α assigns priorities in \mathbb{N} to either states or transitions, and a run is accepting if the highest priority that occurs infinitely often along it is even. An $[i, j]$ -parity automaton, for $i < j$ two natural numbers is a parity automaton whose priorities are in $[i, j]$. A parity automaton is said to be a *weak automaton* if there is no cycle in the automaton containing both an even and odd priority. In a reachability condition, some states are labelled final; a run accepts if it reaches a final state. In a safety automaton, some states are labelled safe; a run accepts if it remains within the safe region.

In a nondeterministic ω -regular automaton $(\Sigma, Q, \iota, \delta, \alpha)$, Σ and Q are finite, and the acceptance condition α is based on the set of states (or transitions) visited infinitely often along a run. A timed automaton (TWA) $(\Sigma, Q, \iota, C, \delta, \alpha)$ has a set of clocks C and its transitions are guarded by inequalities between the clock values and can reset clocks. It reads timed words, which consist of letters of a finite alphabet Σ paired with delays from \mathbb{R} . A timed automaton recognises a timed language, for example “at some point an event occurs twice exactly one time-unit apart.”

We also handle pushdown automata, one-counter automata, vector addition systems with states, one-counter nets and Parikh automata with reachability and safety acceptance. We define these classes of infinite state systems uniformly in Section 5.2 as classes of finite state automata with transitions that modify an infinite *content space*. A visibly pushdown automaton (VPA) is a pushdown automaton without ε -transitions, in which the input alphabet is partitioned into `pop`, `push` and `noop` letters that induce only transitions that `pop`, `push` and have no effect on the stack respectively.

3 History-determinism, simulation and related games

Different simulation-like games that capture either a relationship between automata or properties of a single automaton are at the heart of our technical developments. In this section we go over the various games – both known and newly defined – that will be played throughout this article, and which allow us to connect guidability and history-determinism. These games are all based on Adam (the spoiler) building a word letter by letter, and potentially a run in an automaton over that word, while his opponent Eve (the duplicator) tries to build a single accepting run transition by transition. The differences between these games are based on whether they are played on one or two automata, whether Adam picks transitions, and if so, whether he does it before Eve. The winning condition is similar in all cases: Eve's run must be accepting whenever Adam's run is accepting, or if Adam does not have a run, then whenever the word built by Adam's moves is in the language of a specified automaton. This results in three styles of games: (i) simulation games, played on two automata, in which Adam plays before Eve, and each builds a run in their respective automaton; (ii) token-games,

which can be played on one or two automata, in which Adam first declares the letter, and then Eve plays her transition before Adam plays his; and (iii) the letter game, played on a single automaton, in which Adam only chooses letters and does not build a run at all.

Fair simulation between two LTSs (or automata) is captured by the simulation game defined below:

► **Definition 2** (Simulation game). Consider LTSs $\mathcal{A} = (\Sigma, Q_A, \iota_A, \Delta_A, \alpha_A)$ and $\mathcal{B} = (\Sigma, Q, \iota_B, \Delta_B, \alpha_B)$. The simulation game $\text{Sim}(\mathcal{B}, \mathcal{A})$ between \mathcal{B} and \mathcal{A} is a two player-game played between Adam and Eve with positions in $Q_A \times Q_B$ which starts at the position $(p_0, q_0) = (\iota_A, \iota_B)$. At round i of the play, for $i \geq 0$, when the position is (p_i, q_i) :

- Adam picks $\sigma_i \in \Sigma$ and a transition (or transition sequence, in the presence of ε -transitions) $p_i \xrightarrow{\sigma_i, \rho_i} p_{i+1}$ in \mathcal{A} ;
- Eve picks a transition (or transition sequence, in the presence of ε -transitions) $q_i \xrightarrow{\sigma_i, \rho'_i} q_{i+1}$ in \mathcal{B} ; they proceed from (p_{i+1}, q_{i+1}) .

An infinite play produces a run ρ_A in \mathcal{A} consisting of transitions chosen by Adam and a run ρ_E in \mathcal{B} of transitions chosen by Eve, both on $\sigma_0\sigma_1\sigma_2\cdots$. We say that Eve wins the play if ρ_E is accepting or ρ_A is rejecting.

If Eve has a winning strategy in this game, we say that \mathcal{B} simulates \mathcal{A} , denoted by $\text{Sim}(\mathcal{B}, \mathcal{A})$. It is easy to observe that if \mathcal{B} simulates \mathcal{A} , then $L(\mathcal{A}) \subseteq L(\mathcal{B})$. An LTS \mathcal{A} is *guidable* with respect to a class C of LTSs if \mathcal{A} simulates every LTS \mathcal{A}' in C that satisfies $L(\mathcal{A}') \subseteq L(\mathcal{A})$.

The following *letter game* based definition of history-determinism, was introduced by Henzinger and Piterman [11], and coincides with Colcombet's notion of translation strategies [8].

Given an LTS \mathcal{A} , the *letter game* on \mathcal{A} , denoted by $\text{HD}(\mathcal{A})$ is similar to the simulation game except that instead of playing transitions in an automaton, Adam just chooses letters and builds a word w , letter by letter, which should be in the language of \mathcal{A} , while Eve tries to build a run of \mathcal{A} over w . More precisely, the letter game starts with Eve's token at the initial state ι , and proceeds in rounds. At round i , where Eve's token is at q_i :

- Adam chooses a letter σ_i in the alphabet Σ of \mathcal{A} ;
- Eve chooses a transition $q_i \xrightarrow{\sigma_i} q_{i+1}$ (or a transition sequence $q_i \xrightarrow{\sigma_i, \rho_i} q_{i+1}$ in the presence of ε -transitions) of \mathcal{A} over σ_i ; Eve's token moves to q_{i+1} .

In the limit, a play consists of the word $w = \sigma_0\sigma_1\cdots$ and the run $\rho = \rho_0 \cdot \rho_1 \cdot \rho_2 \cdots$. Eve wins the play if $w \notin L(\mathcal{A})$ or ρ is accepting. We say that \mathcal{A} is history-deterministic (HD) if Eve has a winning strategy in the letter game over \mathcal{A} .

Token games are known to characterise history-determinism on various classes of automata [2, 5, 6]. We generalise token games to be played on two LTSs below, which makes them more akin to a variation of simulation. This will help us relate simulation and history-determinism in Section 4. We only use the 1-token version here.

► **Definition 3** (1-token games over two LTSs (or automata)). Consider LTSs \mathcal{A}' and \mathcal{A} with initial states p_0 and q_0 respectively. In the 1-token game on \mathcal{A}' and \mathcal{A} denoted by $G_1(\mathcal{A}', \mathcal{A})$, Eve has a token with which she constructs a run in \mathcal{A}' , and Adam has a token with which he constructs a run in \mathcal{A} . The game proceeds in rounds, and at round i of the play with token positions (p_i, q_i) , for each $i \geq 0$:

- Adam chooses a letter σ_i in Σ ;
- Eve chooses a transition (or a transition sequence, in the presence of ε -transitions) $p_i \xrightarrow{\sigma_i, \rho'_i} p_{i+1}$ in \mathcal{A}' ;
- Adam chooses a transition (or transition sequence, in the presence of ε -transitions) $q_i \xrightarrow{\sigma_i, \rho_i} q_{i+1}$; the game proceeds from (p_{i+1}, q_{i+1}) .

An infinite play produces a word $w = \sigma_0 \dots$, a sequence of transitions ρ_E of \mathcal{A}' chosen by Eve, and a sequence of transitions ρ_A in \mathcal{A} chosen by Adam. Eve wins if ρ_E is accepting or if ρ_A is rejecting.

A strategy for Eve here is a function $s : (\Delta^+)^* \times \Sigma \rightarrow (\Delta')^*$, where Σ is the alphabet of \mathcal{A} and \mathcal{A}' , and Δ and Δ' are the sets of transitions of \mathcal{A} and \mathcal{A}' , respectively. When clear from context, $G_1(\mathcal{A}', \mathcal{A})$ also denotes the claim that Eve has a winning strategy in the game $G_1(\mathcal{A}', \mathcal{A})$. As an automaton and its induced LTS have the same runs, $G_1(\mathcal{A}', \mathcal{A})$ holds for automata \mathcal{A} and \mathcal{A}' iff it holds for their induced LTSs. We also write $G_1(\mathcal{A})$ for $G_1(\mathcal{A}, \mathcal{A})$.

Note that the 1-token game and the simulation game differ in one key aspect: in the simulation game, Adam plays first, and Eve can use the information of the transition to inform her choice, while in the 1-token game, Eve must choose her transition based only on the *letter* chosen by Adam, who plays his transition after Eve.

4 Criteria for when History-Determinism = Guidability

We now provide criteria which guarantee that history-determinism and guidability coincide for a class of LTSs. In Section 5, we use these to show the coincidence of the two notions for many standard automata classes.

4.1 Closure under (history-)determinism

A first observation is that if every LTS \mathcal{A} can be determinised within the class C , or even if there exists an equivalent HD LTS \mathcal{A}' within C then \mathcal{A} is HD if and only if it is guidable.

► **Lemma 4.** *History-determinism and guidability coincide for any class C of LTSs in which the languages expressed by history-deterministic (or deterministic) LTSs are the same as languages expressed by nondeterministic LTSs.*

The proof is simple: one direction is trivial (HD always implies guidability) and conversely, if an automaton \mathcal{A} is not HD, then it cannot simulate any equivalent HD automaton, implying that \mathcal{A} is not guidable.

Various examples of such classes are provided in Section 5.1, as summarised in Table 1. In particular, the general class of all labelled-transition systems [13], safety/reachability visibly pushdown automata [1], as well as finite-state automata on finite words (NFAs), and co-Büchi, Parity, Rabin, Streett, and Muller automata on infinite words. Yet, this is not the case for Büchi automata or parity automata with a fixed parity index. History-determinism is also strictly less expressive than nondeterminism for pushdown automata, Parikh automata, timed automata and one-counter nets.

4.2 Via token games

For classes that are not closed under determinisation, we have to find some other type of automaton that is difficult to simulate. To do so, we revisit token games, previously used to help decide history-determinism, to relate history-determinism and guidability. Recall that we extended the definition of 1-token games, so that they are played on two automata, rather than one. In the next definition, we use this extended notion of 1-token game to identify, for each automaton \mathcal{A} , an automaton \mathcal{A}' such that Eve wins the the 1-token game on \mathcal{A} if and only if \mathcal{A} simulates \mathcal{A}' .

► **Definition 5** (1-token ghost). *An LTS (or an automaton) \mathcal{A}' is a 1-token ghost of an LTS \mathcal{A} , denoted by $1\text{-TokenGhost}(\mathcal{A}', \mathcal{A})$, if $\mathcal{A}' \equiv \mathcal{A}$ and $G_1(\mathcal{A}', \mathcal{A})$.*

To show that the ghost automaton has the property that $\text{Sim}(\mathcal{A}, \mathcal{A}')$ if and only if Eve wins $G_1(\mathcal{A}, \mathcal{A})$, we compose the strategies in $\text{Sim}(\mathcal{A}, \mathcal{A}')$ and $G_1(\mathcal{A}', \mathcal{A})$.

► **Lemma 6.** *Consider LTSs \mathcal{A} and \mathcal{A}' , such that \mathcal{A} simulates \mathcal{A}' and $1\text{-TokenGhost}(\mathcal{A}', \mathcal{A})$. Then Eve wins $G_1(\mathcal{A})$.*

Proof. Let s_{sim} be a winning strategy of Eve in the simulation game between \mathcal{A} and \mathcal{A}' , and s' her winning strategy in $G_1(\mathcal{A}', \mathcal{A})$. Eve then has a winning strategy s in $G_1(\mathcal{A})$: she plays the strategy s_{sim} in $\text{Sim}(\mathcal{A}, \mathcal{A}')$ against her imaginary friend Berta, who plays the strategy s' in $G_1(\mathcal{A}', \mathcal{A})$ against Adam. In more detail: In each round i of the game $G_1(\mathcal{A})$, Adam chooses a transition sequence ρ_{i-1} in \mathcal{A} on σ_{i-1} (except for the first round) on his token and a letter σ_i , then Berta chooses the transition sequence $\rho_i^B = s'(\rho_0 \dots \rho_{i-1}, \sigma_i)$ over the letter σ_i in \mathcal{A}' on her token in $G_1(\mathcal{A}', \mathcal{A})$, and then Eve chooses the transition sequence $\rho_i = s_{sim}(\rho_0^B \dots \rho_{i-1}^B)$ in \mathcal{A} .

The run built by Eve with the strategy s is accepting if the run built by Berta is, which is in turn accepting if Adam's run is. Hence, s is a winning strategy for Eve in $G_1(\mathcal{A})$. ◀

Then, for classes in which token games characterise history-determinism and which are closed under the ghost relation, guidability and history-determinism coincide.

► **Definition 7.** *A class C of LTSs is closed under 1-token ghost if for every $\mathcal{A} \in C$ there exists $\mathcal{A}' \in C$ such that $1\text{-TokenGhost}(\mathcal{A}', \mathcal{A})$.*

► **Lemma 8.** *Given a class C of LTSs closed under 1-token ghost for which G_1 characterises history-determinism, history-determinism and guidability coincide for C .*

Proof. Being HD always implies guidability, so one direction is easy. For the other direction, if \mathcal{A} simulates every LTS $\mathcal{A}' \in C$, such that $\mathcal{A}' \leq \mathcal{A}$, then in particular it simulates an LTS $\mathcal{A}' \in C$, such that $1\text{-TokenGhost}(\mathcal{A}', \mathcal{A})$, as C is closed under the 1-token ghost. By Lemma 6, Eve wins $G_1(\mathcal{A})$, implying that \mathcal{A} is HD, as G_1 characterises history-determinism in C . ◀

A 1-token ghost is often easy to build, by delaying nondeterministic choices by one letter (Definition 12), as shown in Section 5.2 for pushdown automata, one-counter automata, vector addition system with states, one-counter nets and Parikh automata.

For some automata classes, however, showing closure under 1-token ghosts is trickier: for VPA the stack action must occur as the letter is read, and for timed automata configuration updates are sensitive to the current timestamp. We handle these complications in Section 5.3. We can also only use Lemma 8 with respect to automata classes for which 1-token games characterise history-determinism, which is not the case for parity automata or ω -VPA [2].

4.3 Via Adam's strategy in the letter game

As we will see in detail in Section 5.4, some classes, such as linear automata, are neither closed under 1-token ghost nor determinisation, so there is no hope for the above criteria to apply. Our final criterion is an alternative which, instead of requiring all automata to admit a 1-ghost, builds a difficult-to-simulate automaton from Adam's winning strategy in the letter game. The intuition is that Adam's winning strategy in the letter game on an automaton \mathcal{A} captures behaviour that is difficult for \mathcal{A} to simulate, so if we can turn Adam's strategy into an automaton (which will be language-contained in \mathcal{A} since Adam must play a word in the language of \mathcal{A}), then this automaton will not be simulated by \mathcal{A} . To build this automaton, we first project an automaton \mathcal{B} recognising Adam's winning plays from his strategy onto the alphabet of \mathcal{A} , to obtain an automaton \mathcal{B}_Σ that recognises the words played by Adam's strategy. Then, by taking the 1-token ghost of \mathcal{B}_Σ , we obtain an automaton \mathcal{B}' against which the simulation game is essentially the letter game against Adam's strategy. If the resulting automaton is always still in the class C , guidability and history-determinism coincide.

► **Lemma 9.** *History-determinism and guidability coincide for classes C of LTSs in which, for each $\mathcal{A} \in C$ that is not history-deterministic, there is a deterministic LTS \mathcal{B} over the alphabet of transitions of \mathcal{A} that recognises the plays of a winning strategy of Adam in the letter game on \mathcal{A} , and \mathcal{B} , projected onto the alphabet of \mathcal{A} , has a 1-token ghost in C .*

Proof. Consider a winning strategy τ of Adam in the letter game on \mathcal{A} , and let \mathcal{B} be a deterministic LTS that recognises the plays of τ , seen as runs of \mathcal{A} . Let \mathcal{B}_Σ be the projection of \mathcal{B} onto Σ : it is otherwise like \mathcal{B} , except that its alphabet is Σ instead of the transitions $\Delta_{\mathcal{A}}$ of \mathcal{A} and as a result it has additional nondeterminism. Crucially, every transition in \mathcal{B} is still a transition in \mathcal{B}_Σ . Given a sequence of transitions $t_0 t_1 \dots t_i \in \Delta_{\mathcal{A}}^*$, we call $t''_0 t''_1 \dots t''_i$ its run in \mathcal{B} , which is uniquely defined since \mathcal{B} is deterministic. Note that this sequence of transitions is also a run over the word of $t_0 t_1 \dots t_i$ in \mathcal{B}_Σ . This also extends to infinite sequences. Since every run accepted by \mathcal{B} is a play winning for Adam in the letter game over \mathcal{A} , their projection onto Σ must be in $L(\mathcal{A})$, so $L(\mathcal{B}_\Sigma) \subseteq L(\mathcal{A})$.

Now, let \mathcal{B}' be the 1-token ghost of \mathcal{B}_Σ , witnessed by a strategy s_1 of Eve in the game $G_1(\mathcal{B}', \mathcal{B}_\Sigma)$. Assume, towards contradiction, that $\text{Sim}(\mathcal{A}, \mathcal{B}')$ via some strategy s_{sim} . We construct a strategy s of Eve in the letter game on \mathcal{A} that is winning against τ , in which Eve plays s_{sim} against her imaginary friend Berta in $\text{Sim}(\mathcal{A}, \mathcal{B}')$, who in turn is playing s_1 against Adam in $G_1(\mathcal{B}', \mathcal{B}_\Sigma)$.

In more detail, Adam begins by playing σ_0 according to τ in the letter game on \mathcal{A} ; Berta responds with a transition (or sequence of transitions in the presence of ε -transitions) $\rho'_0 = s_1(\sigma_0)$; and then Eve responds with $s(\sigma_0) = \rho_0 = s_{sim}(\rho'_0)$. On the i^{th} round, when Adam chooses the letter σ_i , after the sequence $\rho_0 \dots \rho_{i-1}$ of Eve's moves in the letter game, and the sequence $\rho''_0, \dots, \rho''_{i-1}$ of transitions in \mathcal{B}_Σ , which is the Σ -projection of the unique run of \mathcal{B} on $\rho_0 \dots \rho_{i-1}$, viewed as a word over $\Delta_{\mathcal{A}}$, Berta makes the move $\rho'_i = s_1(\rho''_0, \dots, \rho''_{i-1}, \sigma_i)$ in $G_1(\mathcal{B}', \mathcal{B}_\Sigma)$, and then Eve the move $s(\sigma_0, \rho_0, \dots, \rho_{i-1}, \sigma_i) = \rho_i = s_{sim}(\rho'_0, \dots, \rho'_{i-1})$ in $\text{Sim}(\mathcal{A}, \mathcal{B}')$ and in the letter game.

We argue that s is winning against τ . Indeed, the run $\rho''_0 \rho''_1 \dots$ in \mathcal{B}_Σ must be accepting since the sequence of transitions $\rho_0 \rho_1 \dots$ that Eve plays agrees with τ . Then, since Berta is playing a winning strategy in $G_1(\mathcal{B}', \mathcal{B}_\Sigma)$, the sequence $\rho'_0 \rho'_1 \dots$ is also an accepting run over the same word. Since Eve is playing a winning strategy in $\text{Sim}(\mathcal{A}, \mathcal{B}')$, the sequence $\rho_0 \rho_1 \dots$ is also an accepting run over the same word. This contradicts τ being a winning strategy for Adam. We conclude that \mathcal{A} does not simulate \mathcal{B}' and is therefore not guidable. ◀

Lemma 9 can be applied to various automata classes, as summarised in Table 1, including ω -regular automata with an $[i, j]$ -parity acceptance condition (Section 5.1), linear automata (Section 5.4), and visibly pushdown automata (Section 5.3).

This concludes our criteria. Concerning the necessity of each criterion, notice that:

- The first criterion (Theorem 1.1) is not subsumed by the others, as demonstrated with the class of all LTSs – it is closed under determinization [7, Theorem 3.4], but G_1 does not characterise history-determinism, which is required for the second criterion, and the letter game need not always be determined, which is required for the third.
- The second criterion (Theorem 1.2) does not imply the first one, as demonstrated by, for instance, safety pushdown automata [10, Theorem 4.1]. The implication from the second criterion to third criterion is unclear, however, and connects to the case of PDA, where the strategies for the players in letter game are not yet understood [10, Section 6].
- Finally, the third criterion (Theorem 1.3) is not subsumed by the other two, as evident from the case of linear automata (Section 5.4).

5 Automata Classes for which History-Determinism = Guidability

5.1 Straightforward cases

By Theorem 1.1, history-determinism and guidability coincide for all automata classes closed under determinisation, including: regular automata (NFAs); VPAs on finite words; ω -regular automata [15]; co-Büchi [16]; and subclasses of ω -regular automata whose deterministic fragment is ω -regular-complete, such as parity, Rabin, Streett, Muller, and Emerson-Lei. Some subclasses of ω -regular automata are not closed under determinisation, e.g., Büchi automata, but as long as they subsume safety automata, we can build on the fact that Adam’s letter-game strategies are recognised by deterministic safety automata, and apply Theorem 1.3: since safety automata are determinisable they are closed under 1-token ghost.

► **Corollary 10.** *History-determinism and guidability coincide for classes of ω -regular automata with an $[i, j]$ -parity acceptance condition, as well as for the class of weak automata.*

5.2 Uniform infinite state systems

In this section, we show that the notions of history-determinism and guidability coincide on the following classes with safety and reachability acceptance conditions: pushdown automata, one-counter automata, vector addition system with states, one-counter nets and Parikh automata. We take a unified approach by defining all of these classes as cases of “uniform automata classes”, and showing that the two notions coincide for such classes (Theorem 14).

These uniform automata classes are specified by a content space \mathcal{C} (e.g., stack contents) and a set \mathcal{K} of partial functions $f : \mathcal{C} \rightarrow \mathcal{C}$ that contains the identity function f_{id} that maps each element in \mathcal{C} to itself (e.g., stack updates). The class specified by \mathcal{C} and \mathcal{K} contains all the automata $\mathcal{A} = (\Sigma, Q, \iota, c_0, \Delta, F_A, F_C)$ that have a finite alphabet Σ , a finite state space Q , and finitely many transitions $(q, \sigma, f, q') \in \Delta$, labelled by a letter $\sigma \in \Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and a function $f \in \mathcal{K}$. The automaton \mathcal{A} induces an LTS that has states $(q, c) \in Q \times \mathcal{C}$, with transitions $(q, c) \xrightarrow{\sigma} (q', c')$, such that (q, σ, f, q') is a transition in \mathcal{A} and $f(c) = c'$.

The acceptance semantics of an automaton in such a class is specified by a set of accepting states $F_A \subseteq Q$ and a set of accepting contents $F_C \subseteq \mathcal{C}$. We will often desire some structure on F_C , so we impose the restriction that F_C belongs to a set $\mathcal{S} \subseteq \mathcal{P}(\mathcal{C})$ of subsets of \mathcal{C} . We call “ $(\mathcal{C}, \mathcal{K}, \mathcal{S})$ -automata” the class of all automata $\mathcal{A} = (\Sigma, Q, \iota, c_0, \Delta, F_A, F_C)$ as above with $F_C \in \mathcal{S}$. Safety automata require an accepting run to have all states in F_A and all content in F_C . We distinguish between synchronous reachability that requires an accepting run to reach an accepting state and an accepting content at the same time, and asynchronous reachability that requires an accepting run to just reach an accepting state and an accepting content, not necessarily at the same time.

► **Definition 11.** *A class of automata is uniform if it can be specified as $(\mathcal{C}, \mathcal{K}, \mathcal{S})$ -automata with either safety, synchronous reachability, or asynchronous reachability acceptance semantic.*

We show that uniform automata classes are closed under 1-token ghost by explicitly constructing for each automaton \mathcal{A} in the class a 1-token ghost, called $\text{Delay}(\mathcal{A})$, inspired by Prakash and Thejaswini [18, Lemma 11]. For each run in \mathcal{A} , we will have a run in $\text{Delay}(\mathcal{A})$ that lags one transition behind. This one-step lag is implemented by storing the previous letter in the state space of \mathcal{A} , in addition to the state of \mathcal{A} ; transitions are then taken based on the previous letter, while reading the current letter, which is now stored.

► **Definition 12** (Delay). Let $\mathcal{A} = (\Sigma, Q, \iota, c_0, \Delta, F_Q, F_C)$ be an automaton in a uniform class $(\mathcal{C}, \mathcal{K}, \mathcal{S})$. The automaton $\text{Delay}(\mathcal{A}) = (\Sigma, Q', \iota', c_0, \Delta', F'_Q, F_C)$ is the Delay of \mathcal{A} , where

1. The set of states Q' is $(Q \times \Sigma) \cup \{\iota'\}$
2. The set of transitions Δ' is given by the union of:
 - $\{(\iota', \sigma, f_{id}, (\iota, \sigma)) \mid \sigma \in \Sigma\}$
 - $\{((q, \sigma), \sigma', f, (q', \sigma')) \mid \sigma, \sigma' \in \Sigma, \text{ and } (q, \sigma, f, q') \in \Delta\}$
 - $\{(q, \sigma), \varepsilon, f, (q', \sigma) \mid \sigma \in \Sigma, \text{ and } (q, \varepsilon, f, q') \in \Delta\}$
3. The set F'_Q consists of state of the form (q, σ) such that $q \in F_Q$, and ι' if $\iota \in F_Q$.

The automaton $\text{Delay}(\mathcal{A})$ has the same acceptance semantics as \mathcal{A} (safety, synchronous reachability or asynchronous reachability).

► **Lemma 13.** Given an automaton \mathcal{A} in a uniform automata class C , the automaton $\text{Delay}(\mathcal{A})$ is in C and is a 1-token ghost of \mathcal{A} .

We show that G_1 characterises history-determinism on all uniform automata classes in the full version [3][Lemma 22], by reducing to safety and reachability LTSs [7]. With Lemma 13 and Theorem 1.2, we get that history-determinism and guidability coincide on all uniform automata classes.

► **Theorem 14.** History-determinism and guidability coincide for uniform automata classes.

It now suffices to represent various automata classes as uniform ones to show that guidability and history-determinism coincide on them. For pushdown and one-counter automata, vector addition systems and one-counter nets, as well as for Parikh automata, the contents are the counter or stack contents, while the update functions are their increments, decrements, pops and pushes. The update partial functions also implement which parts of the contents can be used to enable transitions: for example, for pushdown automata, the partial functions are either defined for all contents where the stack is empty, or undefined for all such contents; for Parikh automata, the contents do not influence which transitions are enabled, so the functions are fully defined. (Formal definitions can be found in the full version [3][Section C.1].)

► **Corollary 15.** History-determinism and guidability coincide for the classes of pushdown automata, one-counter automata, vector addition systems with states, one-counter nets with safety and reachability acceptance conditions, and for Parikh automata with safety, synchronous reachability and asynchronous reachability acceptance conditions.

Non-uniform classes

The class of visibly pushdown automata is not uniform, as there are additional constraints on transitions, namely the kind of function that changes content depends on the letter seen. Timed automata also do not constitute a uniform class, since the alphabet is infinite as it consists of all timed letters, and the clock valuations are updated according to both the transition (resets) and the delay of the input letter. In Section 5.4, we consider linear automata: these are Büchi automata that have no cycles apart from self-loops. Linear automata also does not form a uniform class, since they restrict the state-space. In what follows, we give alternative constructions of 1-token-ghosts for these classes. The case of linear automata is trickier, as we show that it is not closed under 1-token ghost. We therefore use, in Section 5.4, a more involved argument that allows us to use Theorem 1.3.

5.3 Visibly pushdown and timed automata

Visibly pushdown automata over infinite words (ω -VPAs) are neither (history-) determinisable, nor does G_1 characterise history-determinism on them. Nevertheless, we can use Theorem 1.3 to show that history-determinism and guidability coincide for this class.

► **Theorem 16.** *History-determinism and guidability coincide for the class of visibly pushdown automata with any ω -regular acceptance condition.*

Proof sketch. First we show that the class is closed under 1-token ghost. Like in the previous cases, we build an automaton that executes the same transitions, but one step later. The technical challenge is executing transitions with a delay, as an ω -VPA must respect the stack discipline of the input alphabet. We overcome this by maintaining a “semantic stack” that consists of the actual stack and one additional letter that is embedded in the state space and stores, when necessary, the letter that should have been in the top of the stack.

Then, we describe the letter-game for an ω -VPA as a game on a visibly pushdown arena with a “stair parity” acceptance condition, to show that Adam’s winning strategies can be implemented by ω -VPA transducers. We then turn this transducer into a deterministic ω -VPA recognising the plays that agree with Adam’s strategy, and apply Theorem 1.3. ◀

We turn to safety and reachability timed automata, for which we apply Theorem 1.2, yet with a specially tailored Delay construction.

► **Theorem 17.** *For the class of timed automata with safety or reachability acceptance conditions, the notions of history-determinism and guidability coincide.*

Proof sketch. The goal is to simulate such an automaton \mathcal{A} with a delay, as in Definition 12. Yet, the difficulty is that delaying a clock-reset by a step will affect the value of the clock for future comparisons, and there is no delaying of the passage of time. Hence a naive construction would end up recognising the timed language of words in which timestamps are shifted by one. We overcome the difficulty by duplicating in the 1-token ghost construction each clock of \mathcal{A} , using one copy for comparisons in guards and the other to simulate retroactive resets. In addition, the state-space stores the effect of the previous delay, by remembering the corresponding region, that is, how the timestamp compares to existing clocks and constants. With this construction, and the G_1 -characterisation of history-determinism for safety and reachability automata, we complete the proof. ◀

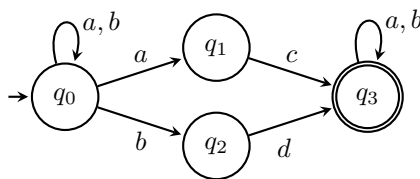
5.4 Linear automata

A *linear* (also called *very weak*) automaton is a Büchi automaton in which all cycles are self loops. (In linear automata, the acceptance condition does not really matter, since over an automaton with only self loops, all the standard ω -regular acceptance conditions coincide.)

First, observe that linear automata are not closed under (history-)determinisation. (The standard Büchi automaton over the alphabet $\Sigma = \{a, b\}$ recognizing the language of finitely many a ’s is linear.) We show that they are also not closed under 1-token ghost, by proving that the linear automaton depicted in Figure 1 admits no 1-token ghost in the class.

► **Theorem 18.** *The class of linear automata is not closed under 1-token ghost.*

Proof. Let \mathcal{A} be the linear automaton depicted in Figure 1, and assume toward contradiction that there is a linear automaton \mathcal{A}' , satisfying $1\text{-TokenGhost}(\mathcal{A}', \mathcal{A})$, witnessed by a winning strategy s of Eve in $G_1(\mathcal{A}', \mathcal{A})$.



■ **Figure 1** A linear automaton which admits no linear automaton that is a 1-token ghost of it.

In a play π_1 of $G_1(\mathcal{A}', \mathcal{A})$ in which Eve plays along s and Adam plays $(ab)^*$ while staying in q_0 , at some points of time $2k-1$ and $2k$, Eve must remain in the same state q' of \mathcal{A}' after Adam chose the letters a and b , respectively, since \mathcal{A}' is linear.

In a play π_2 of $G_1(\mathcal{A}', \mathcal{A})$ in which Eve plays along s and Adam starts with $(ab)^{k-1}a$ while staying in q_0 , Eve reaches, as per the previous claim, the state q' of \mathcal{A}' . Then, if Adam continues with the word ca^ω , while moving from q_0 to q_1 (over the previous a) and then to q_3 (over c), Eve has some accepting continuation run ρ from the state q' over the suffix ca^ω , since s is winning for Eve in $G_1(\mathcal{A}', \mathcal{A})$ and Adam's run is accepting.

Thus, there is an accepting run of \mathcal{A}' on the word $w = (ab)^k ca^\omega$, following in the first $2k$ steps the run of Eve in the play π_1 , reaching the state q' , and then in the next steps following her accepting continuation in π_2 . Yet, \mathcal{A} does not have an accepting run on w , contradicting the equivalence of \mathcal{A} and \mathcal{A}' , and thus the assumption that $\text{1-TokenGhost}(\mathcal{A}', \mathcal{A})$. ◀

Yet, history-determinism and guidability do coincide for the class of linear automata. The underlying reason is that when a linear automaton \mathcal{A} is not history-deterministic, Adam's winning strategy in the letter game can be adapted to a linear automaton that does have a 1-token ghost within the class of linear automata, thus satisfying Theorem 1.3.

▶ **Theorem 19.** *History-determinism and guidability coincide for the class of linear automata.*

Proof sketch. History-determinism implies guidability with respect to all classes. For the other direction, consider a linear automaton $\mathcal{A} = (\Sigma, Q, \iota, \Delta, \alpha)$ that is not HD, and let $\mathcal{M} = (\Delta, \Sigma, M, m_0, \Delta_M : M \times \Delta \rightarrow M, \gamma : M \rightarrow \Sigma)$ be a deterministic finite-state transducer representing a finite-memory winning strategy s_M of Adam in the letter game.

We then build, by taking a product of \mathcal{M} and \mathcal{A} , a deterministic safety automaton \mathcal{P} , that recognises the set of plays that can occur in the letter game on \mathcal{A} if Adam plays according to s_M . From \mathcal{P} , we take its projection \mathcal{N} onto the alphabet Σ of \mathcal{A} . \mathcal{N} need not be linear, but we adapt it into a linear \mathcal{N}' that will still correspond to a winning strategy of Adam in the letter game. \mathcal{N}' will thus constitute a projection of a deterministic automaton \mathcal{P}' onto the alphabet Σ , where \mathcal{P}' is over the alphabet of transitions of \mathcal{A} and recognises the plays of a winning strategy of Adam in the letter game. Once achieving that, we can apply the Delay construction on \mathcal{N}' – it will not introduce, in this case, non-self cycles, since the states of \mathcal{N}' (as the projection of the states of \mathcal{P}'), have outgoing transitions only on a single letter. Hence, we satisfy Theorem 1.3, proving the stated claim. ◀

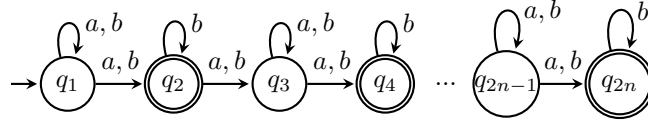
6 Automata Classes for which History-Determinism \neq Guidability

In this section we study classes which admit guidable automata that are not history-deterministic. They offer insight into how, in practice, the criteria can fail to hold, and witness that even on arguably natural automata classes, guidability and history determinism do not necessarily coincide. The main reason for the equivalence between the notions to fail for these classes is a bound on the allowed resources – the number of states in the first class and the number of clocks in the second.

12:14 History-Determinism vs Fair Simulation

Our first example of when history-determinism and guidability differ are Büchi automata with a bounded number of states, witnessed by the automaton in Figure 2.

► **Theorem 20.** *For every $n \in \mathbb{N}^+$, history-determinism and guidability are distinct notions for the class of Büchi automata with up to $2n$ states.*



■ **Figure 2** A Büchi automaton that accepts words with a finite number of a 's. To simulate any equivalent small enough Büchi automaton \mathcal{B} , Eve moves to the next accepting state once the other automaton is in a maximally strongly connected component with an accepting state. The size constraint on \mathcal{B} , and the observation that a such a component can not both have a transition on a and an accepting state guarantees that this strategy wins in the simulation game. However, \mathcal{B} is not history-deterministic.

This counter-example is simple, but quite artificial. We proceed with a class which is, arguably, more natural: timed automata with a bounded number of clocks.

► **Theorem 21.** *History-determinism and guidability are distinct notions for the class \mathbb{T}_k of timed-automata over finite words with at most k clocks, for each $k \in \mathbb{N}$.*

Proof sketch. We consider the language of infinite words in which there are k event pairs that occur exactly one time-unit apart both before and after the first occurrence of a $\$$ letter. Then, the guidable automaton for this language can freely reset its clocks until the $\$$ -separator, which allows it to ensure it tracks all delays tracked by a smaller automaton with up to k clocks. Crucially, any automaton that only accepts words in this language must keep track of k clock values when the separator occurs, as otherwise, it will also accept some word in which the second of matching pair of event is shifted a little. ◀

7 Conclusions

We have presented sufficient conditions for a class of automata to guarantee the coincidence of history-determinism and guidability, and used them to show that this is the case for many standard automata classes on infinite words. As a result, we get algorithms to decide guidability for many of these classes. Guidable automata allow for simple model-checking procedures, and once guidability check is simple, one can take advantage of it whenever applicable. For example, consider a specification modelled by a Büchi or coBüchi automaton \mathcal{A} . Model-checking whether a system \mathcal{S} satisfies \mathcal{A} is *PSPACE*-hard. Using our results, one can check first in *PTime* whether \mathcal{A} is guidable, and in the fortunate cases that it is, conclude the model checking in *PTime*, by checking whether \mathcal{A} simulates \mathcal{S} . We have also demonstrated automata classes for which guidability and history-determinism do not coincide.

We believe that our positive results extend to additional automata classes, such as register automata [14], which behave quite similarly to timed automata. Furthermore, we believe them to extend to additional families of automata classes:

- *Finite words.* We have focused on automata over infinite words, which in this context, are better behaved. Ends of words bring additional complications to our constructions, but overall we believe our approach to be amenable to the analysis of finite word automata.

- *Quantitative automata.* In quantitative automata, transitions carry additional information in the form of weights. As a result, there is an additional difference between the letter game and simulation game, which makes extending our analysis to the quantitative setting particularly relevant. We believe that many of our techniques adapt to that setting.

One could argue that for model-checking, the more interesting property is whether a (not necessarily safety) automaton is guidable by just safety automata, since we typically represent specifications by safety automata. Interestingly, this property often coincides with guidability w.r.t. the full class of automata, as demonstrated in by our third criterion (Theorem 1.3): if Adam’s strategies in the letter game can be translated into automata, these automata are safety ones, and therefore guidability w.r.t. safety automata is just as hard as guidability w.r.t. the full class of automata with the more complex acceptance conditions.

References

- 1 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.
- 2 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)*, page 16, 2018.
- 3 Udi Boker, Thomas A. Henzinger, Karoliina Lehtinen, and Aditya Prakash. History-determinism vs fair simulation, 2024. arXiv:2407.08620.
- 4 Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak. On succinctness and recognisability of alternating good-for-games automata. *arXiv preprint*, 2020. arXiv:2002.07278.
- 5 Udi Boker and Karoliina Lehtinen. Token games and history-deterministic quantitative automata. In *FoSSaCS*, pages 120–139, 2022. A submitted journal version is available at arXiv:2110.14308.
- 6 Udi Boker and Karoliina Lehtinen. When a little nondeterminism goes a long way: An introduction to history-determinism. *ACM SIGLOG News*, 10(1):24–51, 2023. doi:10.1145/3584676.3584682.
- 7 Sougata Bose, Thomas A. Henzinger, Karoliina Lehtinen, Sven Schewe, and Patrick Totzke. History-deterministic timed automata, 2023. arXiv:2304.03183.
- 8 Thomas Colcombet. Fonctions régulières de coût. *Habilitation à diriger les recherches, École Doctorale de Sciences Mathématiques de Paris Centre*, 2013.
- 9 Thomas Colcombet and Christof Löding. The non-deterministic mostowski hierarchy and distance-parity automata. In *Proc. of ICALP*, volume 5126, pages 398–409, 2008. doi:10.1007/978-3-540-70583-3_33.
- 10 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A bit of nondeterminism makes pushdown automata expressive and succinct. *Log. Methods Comput. Sci.*, 20(1), 2024. doi:10.46298/LMCS-20(1:3)2024.
- 11 Thomas Henzinger and Nir Piterman. Solving games without determinization. In *Proceedings of CSL*, pages 395–410, 2006.
- 12 Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. Fair simulation. *Inf. Comput.*, 173(1):64–81, 2002. doi:10.1006/inco.2001.3085.
- 13 Thomas A. Henzinger, Karoliina Lehtinen, and Patrick Totzke. History-deterministic timed automata. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPICs*, pages 14:1–14:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

12:16 History-Determinism vs Fair Simulation

- 14 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.
- 15 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.
- 16 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω -words. *Theoretical Computer Science*, 32:321–330, 1984.
- 17 Damian Niwiński and Michał Skrzypczak. On Guidable Index of Tree Automata. 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 81:1–81:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2021.81.
- 18 Aditya Prakash and K. S. Thejaswini. On history-deterministic one-counter nets. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 218–239. Springer, 2023. doi:10.1007/978-3-031-30829-1_11.