When a Little Nondeterminism Goes a Long Way: An Introduction to History-Determinism

Udi Boker, Karoliina Lehtinen

When a Little Nondeterminism Goes a Long Way: an Introduction to History-Determinism

Udi Boker, Reichman University, Herzliya, Israel
Karoliina Lehtinen, CNRS, Aix-Marseille Université, LIS, Marseille, France

History-deterministic automata are an intermediate automata model, in between deterministic and nondeterministic ones. An automaton is history-deterministic if its nondeterminism can be resolved on-the-fly, by only taking into account the prefix of the word read so far. This restricted form of nondeterminism yields a class of automata that retains some of the algorithmic properties of deterministic automata, while also enjoying some of the power of nondeterminism.

History-determinism has received a lot of attention in the past few years and this article surveys some of the recent developments on the topic. It aims to showcase some of the key constructions and techniques involved in this line of research, while remaining fairly informal.

We begin with a roughly chronological overview of research on history-determinism, intended as a quick reference to some of the main results on the subject. We then move onto a more detailed discussion of what are, in our view, the key aspects of history-determinism, highlighting throughout questions that remain open. Like any survey written on a topic that is actively worked on, this article aspires to be out-of-date as soon as it is published. We hope that it will remain useful as a gentle, if incomplete, introduction to the topic.

Additional Key Words and Phrases: history-determinism, good-for-games automata, nondeterminism

1. A BRIEF HISTORY OF HISTORY-DETERMINISM

Most nondeterministic automata models are more expressive or more succinct than their deterministic counterparts; however, this comes at a cost, as deterministic automata tend to have better algorithmic properties. Intermediate models that allow some restricted forms of nondeterminism aim to combine some of the algorithmic benefits of determinism with the increased power of nondeterminism, thus enjoying (some of) the best of both worlds. History-determinism is one such intermediate model.

History-determinism, as it is understood today, has its roots in several independently invented notions. Kupferman, Safra and Vardi studied, already in 1996 [Kupferman et al.(2006)], languages of tree automata derived from word automata, leading to a notion that has been later referred to as automata that are *good-for-trees* [Boker et al.(2013)]. Independently, in 2006, Henzinger and Piterman observed that an automaton need not be deterministic for it to be usable in the classical algorithm to solve games with ω-regular winning conditions [Henzinger and Piterman(2006)], which is a key part of solving the reactive synthesis problem, known as Church synthesis. This algorithm consists of taking the product of the game to be solved with a deterministic automaton recognising its winning condition in order to reduce it to a game with a standard winning condition, such as a parity game. Henzinger and Piterman's insight was that *some* nondeterministic automata, which they called *good-for-games*, have sufficient compositional properties for this reduction to work without the need to determinise them first. Finally, Colcombet introduced the term history-determinism in his work on regular cost-automata [Colcombet(2009)]. In this setting, nondeterministic automata are strictly more expressive than deterministic ones, and history-deterministic automata provide a useful intermediate model, as they express the same functions as nondeterministic ones, while allowing for better algorithmic manipulations, in particular thanks to their compositional properties.

Since then, these classes of automata have received their fair share of attention, not least because of their relevance to reactive synthesis. Initially, they were mainly studied in the ω-regular setting, where, despite differences in their definitions, the notions of good-for-trees, good-for-games and history-determinism are equivalent [Boker and

Lehtinen(2019)]. Although the latter two terms have been used largely interchangeably, the two definitions do not coincide in all settings, as we shall discuss further down; therefore, in this survey, we adopt the term history-determinism, which best captures the intuition of the definitions we use. Indeed, we say that an automaton is history-deterministic if its nondeterminism can be resolved on-the-fly, based only on the history of the input word, without knowledge of the suffix still to be read. This is in contrast to general nondeterministic automata, in which nondeterminism is often resolved by *guessing* something about the future of the word (See Fig. 1). We call the function witnessing history-determinism a *resolver*.

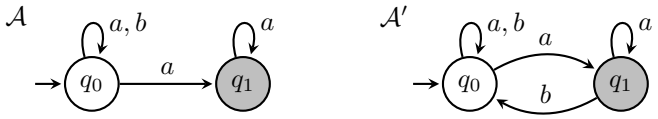

Fig. 1. Nondeterministic coBüchi automata that recognise the language of ω-words with a finite number of b's. A run is accepting if it eventually remains in the accepting state q_1. $\mathcal{A}$ is not history-deterministic as it needs to move to q_1 after the last b has been seen—information which depends on the future, rather than the history, of the input. $\mathcal{A}'$ on the other hand is history-deterministic: to build an accepting run on any word in the language, it suffices to eventually choose, *whenever* the automaton is in state q_0, the transition into q_1.

Initially, it was not clear whether this form of nondeterminism brought much to the table. Indeed, the observant reader might have noticed that the history-deterministic automaton $\mathcal{A}'$ in Fig. 1 is just a deterministic automaton with an additional transition that does not affect the language of the automaton; it is *determinisable by pruning* (DBP) [Aminof et al.(2010); Boker et al.(2013)]. It took until 2013 for Boker, Kuperberg, Kupferman and Skrzypczak to establish examples of history-deterministic automata that are not DBP [Boker et al.(2013)], one of which is reproduced in Fig. 2.

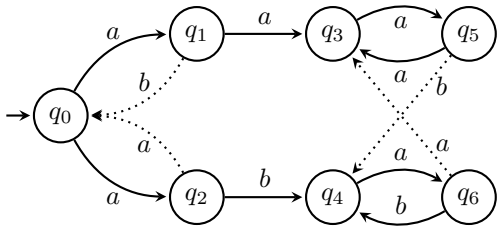

Fig. 2. A history-deterministic coBüchi automaton from [Boker et al.(2013)], where dotted transitions are rejecting, that is not DBP. It recognises the language $(aa + ab)^*[a^\omega + (ab)^\omega]$. From q_0, the automaton must guess whether the next letter is an a or a b in order to access the second part of the automaton, which is deterministic. It is not DBP, since pruning the transition from q_0 to q_1 does not allow to accept a^ω, and pruning the transition from q_0 to q_2 does not allow to accept $(ab)^\omega$. At first glance, it also does not seem to allow for a history-deterministic behaviour. However, any word that is in the language must eventually settle on whether the letters at even positions are a's or b's. Using this fact, the resolver can, from the initial state, alternate between guessing that the next letter is a (choosing q_1) and b (choosing q_2). Then, if the resolver is always wrong in its guess, the word is not in the language, while on any word in the language, the resolver succcessfully builds an accepting run. For a Büchi example, see [Boker et al.(2013), Fig. 2].

Despite not being DBP, the automaton in Fig. 2 is still equivalent to a deterministic automaton of the same size. Colcombet conjectured in his habilitation that this is always the case for history-deterministic ω-regular automata [Colcombet(2013), Conjecture 7.27]. In 2015, Kuperberg and Skrzypczak disprove this, showing that history-deterministic (coBüchi) automata can be exponentially more succinct than their deterministic counterparts [Kuperberg and Skrzypczak(2015)] (see Section 5). With this key

insight, history-deterministic automata became very interesting indeed: they enable, for some specifications at least, the synthesis problem to be solved with the complexity of solving parity games—quasipolynomial at the moment [Calude et al.(2017)]—bypassing an expensive and difficult determinisation step that leads to an inevitable exponential blow-up.

The notion of history-determinism generalises naturally to alternating automata [Colcombet(2013); Boker and Lehtinen(2019)], by requiring that both the nondeterministic choices of an automaton, and the dual, universal choices, are history-deterministic. Similarly to the nondeterministic setting, history-deterministic alternating parity automata enjoy all the same compositional properties as their nondeterministic counterparts and can also be used directly to solve games. They can be exponentially more succinct than nondeterministic parity automata, but also enjoy a single-exponential determinisation procedure [Boker et al.(2020b)]. An alternating automaton can also be *half* history-deterministic, if only its nondeterministic (but not universal) choices are history-deterministic. Although this class of automata does not enjoy all of the compositional properties of fully history-deterministic automata, some problems, such as model-checking whether all executions of a system are in the language of the automaton, are no harder for this class than for deterministic automata. Unfortunately, recognising half history-deterministic alternating automata is PSPACE-hard [Boker et al.(2020b)].

History-deterministic ω-regular automata share additional useful properties with deterministic ones, some of which relate to *typeness* of automata, namely to the existence of automata of different types over the same structure recognising the same language [Boker et al.(2017)]. Recently, they have been further shown to have properties related to minimality that make them appealing as an alternative for deterministic ones. Radi and Kupferman showed that transition-based history-deterministic coBüchi automata enjoy PTIME minimisation [Abu Radi and Kupferman(2022)]; in contrast, the problem is NP-complete for deterministic automata and state-based history-deterministic ones [Schewe(2020)]. Furthermore, Casares, Colcombet and Lehtinen showed that the size of a minimal transition based history-deterministic Rabin automaton recognising a Muller language L coincides with the size of the memory that winning strategies need in games with winning condition L [Casares et al.(2022)]; in contrast, the size of a minimal deterministic automaton coincides with the size of the *chromatic memory* needed in these games, that is, the size of a memory that only sees the labels of the game, rather than the states visited [Casares(2022)]. These results highlight the fundamental nature of history-determinism, on par, in many ways, with deterministic automata.

While the early work on history-determinism focused on ω-regular automata and regular cost functions, the differences between determinism and nondeterminism are even more significant with more general models. For pushdown automata, for example, nondeterminism is more expressive than determinism, and for languages recognised by deterministic pushdown automata, nondeterministic ones can be arbitrarily more succinct [Hartmanis(1980); Valiant(1976)]. On the algorithmic side, however, problems such as universality and reactive synthesis are undecidable for nondetermistic pushdown automata, while they are EXPTIME-complete for deterministic ones [Hopcroft et al.(2007)]. Intermediate models, such as history-determinism, have the potential to combine some of the best of both worlds. Lehtinen and Zimmermann first showed that history-deterministic ω-pushdown automata are indeed more expressive than deterministic ones, while retaining the EXPTIME decidability of synthesis and universality [Lehtinen and Zimmermann(2022)]. With Guha and Jecker, they then showed that this is already true on finite words, and that futhermore, history-deterministic pushdown automata are at least exponentially more succinct than their deterministic coun-

terparts [Guha et al.(2021)]. While history-deterministic pushdown automata have poor closure properties, history-deterministic visibly pushdown automata, which are exponentially more succinct but not more expressive than deterministic ones, somewhat mitigate these issues: they are closed under intersection, union and complement, and, unlike for general pushdown automata, the problem of deciding whether a visibly pushdown automaton is history-deterministic is decidable, and even in EXPTIME.

Another setting in which nondeterminism is more expressive than determinism is the quantitative one [Chatterjee et al.(2010)]. There, automata no longer recognise languages, but rather, they define functions from words to values, and enable the modeling of various non-Boolean aspects of the world. Boker and Lehtinen generalised the definitions of history-determinism and good-for-gameness to the quantitative setting, and observed that unlike in the ω-regular setting, these two notions no longer coincide: while history-determinism implies good-for-gameness, the converse is not true [Boker and Lehtinen(2021)]. In addition to these two distinct notions, in the quantitative setting it also makes sense to consider notions of threshold history-determinism and threshold determinisability by pruning, and even approximative history-determinism. Whether some of these notions coincide depends largely on the type of quantitative automata considered (see Figure 2 in [Boker and Lehtinen(2021)]).

Unfortunately, recognising history-determinism is not always easy. Henzinger and Piterman proposed an EXPTIME algorithm [Henzinger and Piterman(2006)], based on checking whether an automaton simulates an equivalent deterministic one. The cost of determinisation dominates the complexity of this procedure. Kuperberg and Skrzypczak proposed a notion of *joker game*, which they used to give a polynomial decision procedure to check the history-determinism of coBüchi automata [Kuperberg and Skrzypczak(2015)]. Bagnol and Kuperberg later extended the *one-token game* of [Löding and Repke(2013)] into a *two-token game*, which, they showed, characterises history-determinism for Büchi automata [Bagnol and Kuperberg(2018)] and is solvable in PTIME. Later, Boker, Kuperberg, Lehtinen and Skrzypczak showed that the same algorithm also works for coBüchi automata [Boker et al.(2020a)]. Bagnol and Kuperberg conjectured that it is also correct for parity automata, and therefore for all ω-regular automata, a conjecture that remains open to this day.

As in the Boolean setting, the problem of deciding whether a quantitative automaton is history-deterministic is non-trivial. It is particularly relevant, as it is polynomially equivalent to a variant of the quantitative synthesis problem called best-value synthesis (see Section 7.3). Boker and Lehtinen extended token games to the quantitative setting, and showed that the one-token game can be used to decide history-determinism of all quantitative automata on finite words, Safety and Reachability automata on infinite words, as well as some quantitative automata on infinite words, specifically discounted-sum and Inf automata [Boker and Lehtinen(2022)]. They further showed that the two-token game can also be used to decide the history-determinism of LimSup, LimInf and Sup automata on infinite words.

The study of history-determinism continues, and extends to other models, such as timed [Henzinger et al.(2022); Bose et al.(2022)] and Parikh automata [Erlich et al.(2022)], as well as one-counter nets [Prakash and Thejaswini(2022)]. Exploring how it can be exploited in specification logics, for instance in fragments of LTL [Iosti and Kuperberg(2019)], is another promising direction. Yet many fundamental questions remain open, even in the Boolean setting. We will highlight some of these throughout this survey. In the rest of this article, we aim to introduce some of the key notions and techniques that appear in the body of work on history-determinism, as well as our favourite examples of history-deterministic automata. We will keep our discussion fairly informal, emphasising the core ideas while referring the reader to the

original material for more precise definitions and proofs. A more technical exploration of some of the material can also be found in [Kupferman(2023)].

We start, in Section 2, with the definition of history-determinism, as originally provided for ω-regular automata, and as extended to other automata models. We continue, in Section 3, with the similarity and differences between history-determinism and other notions, such as good-for-gameness. In Sections 4 and 5 we consider the expressiveness and succinctness, respectively, of history-deterministic automata compared to deterministic and nondeterministic ones. In Section 6 we describe the current techniques, which are mainly based on token games, used for deciding whether a given automaton is history-deterministic. In Section 7 we describe some applications of history-determinism in model checking and synthesis. In Section 8 we discuss the nature of *resolvers*, the strategies used for resolving nondeterminism in history-deterministic automata, and wrap up in Section 9 with some concluding remarks.

2. DEFINING HISTORY-DETERMINISM

We start with the basic definition of history-determinism, as was originally provided for nondeterministic ω-regular automata, and continue with a discussion of how history-determinism adapts, or not, to more involved automata models. In the quantitative setting, we also mention threshold- and approximative-history-determinism. To keep our discussion of this large variety of models manageable, we only give partially formal definitions.

2.1. The basic definition of history-determinism for nondeterministic (ω-)regular automata

A nondeterministic (ω)-regular automaton is a structure $\mathcal{A} = (\Sigma, Q, I, \delta, \alpha)$, where Σ is an alphabet; Q is a finite nonempty set of states; $I \subseteq Q$ is a set of initial states; $\delta \colon Q \times \Sigma \to 2^Q$ is a transition function, and α is an acceptance condition. An automaton is deterministic if I is a singleton, and for every state q and letter σ, we have $|\delta(q, \sigma)| \leq 1$. A run of the automaton is a path over its states (or transitions), starting with an initial state and continuing along the transition function. A run is accepting if it satisfies the acceptance condition; a word is accepted by $\mathcal{A}$ if there is an accepting run on it; and the language that $\mathcal{A}$ recognises, denoted by $L(\mathcal{A})$, is the set of words that it accepts. Two automata $\mathcal{A}$ and $\mathcal{A}'$ are equivalent, denoted by $\mathcal{A} \equiv \mathcal{A}'$, if they recognise the same language. In the case of regular automata, the acceptance condition α is a subset of Q, and a run is accepting if it ends in a state in α. In the case of ω-regular automata, there are various acceptance conditions for whether an infinite run is accepting, among which are Büchi, coBüchi, parity, and many more. (More on acceptance conditions of ω-regular automata can be found, for example, in [Boker(2018)].)

We provide the following definition of history-determinism, based on a *letter game*, which was introduced by Henzinger and Piterman [Henzinger and Piterman(2006)], but also coincides with Colcombet's notion of translation strategies [Colcombet(2013)].

Given an ω-regular automaton $\mathcal{A}$, the *letter game* on $\mathcal{A}$ is a two-player win-lose game between Adam and Eve in which Adam builds a word w, letter by letter, and Eve builds a run of $\mathcal{A}$ over w, transition by transition. More precisely, at turn i:

— Adam chooses a letter a_i in the alphabet Σ of $\mathcal{A}$;
— Eve chooses a transition τ_i of $\mathcal{A}$ over a_i.

In the limit, a play consists of the word $w = a_0 a_1 \dots$ and the sequence of transitions $\rho = \tau_0 \tau_1 \dots$. Eve wins the play if either $w \notin L(\mathcal{A})$ or ρ is an accepting run of $\mathcal{A}$ over w. We say that $\mathcal{A}$ is history-deterministic (HD) if Eve has a winning strategy in the letter game over $\mathcal{A}$, which we denote by $\mathsf{HD}(\mathcal{A})$.

In the case that $\mathcal{A}$ is regular, namely runs on finite words, the winning condition is a safety condition: at the end of each turn, if the word built so far is in $L(\mathcal{A})$, the run

built so far must be an accepting run of $\mathcal{A}$ over it. Equivalently, Adam has the choice to end a play at any turn to check the winning condition.

Since many different games and strategies will feature in this article, we differentiate the strategy that witnesses history-determinism by calling it a *resolver*. For a less anthropomorphic definition, a resolver can also be seen simply as a function $r : \Sigma^* \times \Sigma \to \Delta$, where Δ is the set of transitions, that induces a run over every word w, such that the induced run is an accepting run of $\mathcal{A}$ over w whenever the word w is in $L(\mathcal{A})$. An automaton is then history-deterministic if it has a resolver. For more involved automata models, discussed below, the history will need to contain more than just the prefix of the word read so far, so in general we view the resolver as a function $r : \Delta^* \times \Sigma \to \Delta$. In the presence of ε-transitions, Eve must be allowed to play not just one transition, but a sequence of transitions in the letter-game.

2.2. External memory and extended input, transitions and acceptance condition

As long as the input is ordered, in the sense of having the "history" of what has been read and the "future" of what is to be read (as opposed to two-way automata for example), and all of the run information is captured by the transitions, the definition of history-determinism, based on the letter game, is generally the same as in Section 2.1.

For example, having an input over an infinite alphabet (like timed [Henzinger et al.(2022)] or register automata) or having infinitely many states (like a labelled transition system [Henzinger et al.(2022)]) makes no difference; in pushdown or counter automata, the transition history keeps track of changes in the counter or stack, providing the resolver with this extra information [Guha et al.(2021), Section 3]; and in Parikh automata, the transition history contains the traversed number vectors, while the more involved acceptance condition follows as is to the letter game [Erlich et al.(2022)].

On more complex inputs, such as trees, graphs, and other structures, the definition of "history" and "future" is not as straightforward. We are not aware of existing definitions of history-determinism in these cases. For tree automata, if the path from the root to the current node is the history, and Adam is allowed to choose both the next letter and the next child in the letter game, one gets a definition that relates to good-for-trees automata (cf. [Kupferman et al.(2006); Boker et al.(2013)]). Another approach is the notion of *guidable automata* [Colcombet and Löding(2008)], which are automata that can simulate (with a positional strategy) any automaton for the same, or smaller, language.

Regarding two-way automata, to our understanding history-determinism does not fit such models, as there is no sense of "history" and "future" in the word (at least on finite words), which can be read back and forth.

2.3. Alternation

As specification formalisms, alternating automata are closer to logics, as they allow both nondeterministic choices (that correspond to logical disjunctions) and universal choices (that correspond to logical conjunctions).

It turns out that history-determinism extends quite naturally to alternating automata [Colcombet(2013); Boker and Lehtinen(2019)]. The definition of the letter game can be adjusted so that Adam is in charge of resolving universal choices, allowing him to try sabotaging Eve's attempt to build an accepting run. Then, the existence of a winning strategy for Eve determines whether the *nondeterminism* in an alternating automaton is history-deterministic. However, this asymmetrical definition, which we call half-history-determinism, only preserves the winner of one-player games. To regain full compositionality, we must ask also for the dual: the universal choices must also be history-deterministic. This can be defined by requiring Adam to have a win-

ning strategy in the dual letter game, in which Eve plays letters in order to build a word that is not in the language of the automaton, and Adam attempts to resolve universal choices to build a rejecting run transition by transition, while Eve gets to resolve nondeterministic choices. Since alternating automata are easy to complement syntactically, this is equivalent to the *nondeterminism* in the syntactic complement automaton being history-deterministic. If both the nondeterminism and universality are history-deterministic, we call the alternating automaton history-deterministic, and this ensures its compositional properties with respect to both games and other alternating automata [Colcombet(2013); Boker and Lehtinen(2019)].

While defining history-determinism for alternating ω-regular automata is pleasingly natural, it might get trickier for more complex alternating models. For example, ε-transitions tend to make alternating automata asymmetric (is an infinite ε-sequence accepting or rejecting?); this breaks the duality between nondeterminism and universality, and affects possible definitions of history-determinism.

2.4. Quantitative setting

A quantitative[1] automaton $\mathcal{A}$ extends a Boolean one by having real weights on transitions and a value function $\mathsf{Val} : \mathbb{R}^* \to \mathbb{R}$ or $\mathsf{Val} : \mathbb{R}^\omega \to \mathbb{R}$, which assigns real values to runs instead of acceptance/rejection [Chatterjee et al.(2010)]. It is then called a Val-automaton (e.g., a Sum-automaton). The value $\mathcal{A}(w)$ of $\mathcal{A}$ on a word w is the supremum (or infimum) of $\mathsf{Val}(\pi)$ over all runs π of $\mathcal{A}$ on w. Automata $\mathcal{A}$ and $\mathcal{A}'$ are equivalent if they realise the same function from words to real numbers. Various value functions are used in the literature, among which are $\mathsf{Sum}, \mathsf{LimInf}, \mathsf{LimSup}$, and many others.

History-determinism. To adapt the letter game to the quantitative setting [Boker and Lehtinen(2021)], it suffices to adjust its winning condition: instead of Eve having to build an accepting run, she now has to build an *optimal* run that has the value of $\mathcal{A}$ on the word built by Adam. One immediate consequence of this is that, since the value of a word is the supremum among its runs, it might be the case that an automaton is not history-deterministic simply because there is no run with the value of the word. For alternating automata, a similar extension applies also to the dual letter game, in which Adam now has to build a non-optimal run.

Threshold history-determinism. Another, related but different, generalisation of history-determinism to the quantitative setting is the following. An automaton is *threshold history-deterministic* [Boker and Lehtinen(2021)] if for all threshold values t, Eve has a strategy in the *threshold letter game*, in which she only needs to guarantee that if the word built by Adam has value at least t, then the run built by Eve also has value at least t. In other words, the automaton can have different resolvers for each threshold value. The implication in the winning condition is reminiscent of the Boolean case; indeed, this notion corresponds to all the Boolean automata derived by adding a threshold acceptance condition to the quantitative automaton (i.e., a run is accepting if it has value at least t) being history-deterministic, in the Boolean sense.

While every history-deterministic automaton is also threshold history-deterministic, the converse is generally false, as demonstrated in Fig. 3. Observe that this counterexample only requires three distinct values, and can easily be implemented with virtually any value function.

Approximation. Moving from Boolean to quantitative automata allows for approximate solutions, which are important in areas such as verification, with inherently hard

[1] We follow the distinction suggested in [Boker(2021)] between "quantitative"and "weighted" automata.

problems. Intuitively, an automaton is approximatively history-deterministic if it has a resolver that results in runs that are optimal up to the required approximation.

Various works on approximative history-determinism appear in the literature, among which are Colcombet's original definition of history-determinism [Colcombet(2009)] with respect to cost-function automata (more on history-determinism in the setting of cost-function automata can be found in [Colcombet and Löding(2010); Colcombet and Fijalkow(2016)]), results in [Hunter et al.(2016); Hunter et al.(2017); Filiot et al.(2017)], where the approximative notion of history-determinism is called r-GFGness, and results in [Aminof et al.(2010); Hunter et al.(2016)], where the notion lies somewhere between approximative determinisability by pruning and approximative history-determinism.

Weighted automata. As opposed to quantitative automata, discussed above, a weighted automaton is defined with respect to a semiring (or more generally with respect to an (ω-)valuation monoid), and multiple transitions over the same input letter from the same state need not be interpreted as a nondeterministic choice; their interpretation is given by the semiring-summation operation. It therefore seems that history-determinism does not fit general weighted automata.

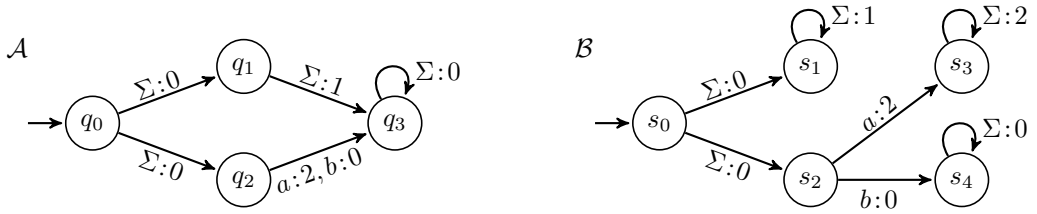

Fig. 3. Automata that are threshold-history-deterministic and good-for-games, but not history-deterministic [Boker and Lehtinen(2021)]. The automaton $\mathcal{A}$ can be seen, for example, as a Sum/DSum/Sup-automaton, and $\mathcal{B}$ as a Avg/LimSup/LimInf/LimSupAvg/LimInfAvg-automaton. Taking $\mathcal{A}$ as a Sum automaton, for instance, it is not history-deterministic, since if the nondeterminism in q_0 is resolved by going to q_1, the resulting automaton is not equivalent to $\mathcal{A}$ with respect to the word a^∞, and if it is resolved by going to q_2, the resulting automaton fails on ab^∞. On the other hand, $\mathcal{A}$ is threshold history-deterministic: For a threshold up to 3, the nondeterminism is resolved by going to q_1 and for a threshold above 3 by going to q_2.

3. HISTORY-DETERMINISM VS. GOOD-FOR-GAMENESS AND OTHER NOTIONS

History-determinism is related to various other notions, the best-known of which is good-for-gameness. Among the others are determinisabiltiy-by-pruning and good-for-"something", where this something can be automata, trees, MDPs, and other entities.

An automaton $\mathcal{A}$ is determinisable-by-pruning if we can obtain an equivalent deterministic automaton by removing some of $\mathcal{A}$'s transitions [Aminof et al.(2010); Boker et al.(2013)].

An automaton $\mathcal{A}$ is good for some entities E, e.g., games or Markov decision processes (MDPs), if the composition of $\mathcal{A}$ with every entity E whose definition is based on the language or function that $\mathcal{A}$ realises yields an entity $E \times \mathcal{A}$ that is equivalent to E.

For example, a Boolean automaton $\mathcal{A}$ over the alphabet Σ is good-for-games if for every win-lose game G with Σ-labeled transitions and winning condition $L(\mathcal{A})$, we have that G and $G \times \mathcal{A}$ have the same winner [Henzinger and Piterman(2006); Boker and Lehtinen(2019)]. When $\mathcal{A}$ is quantitative, we require that for every zero-sum game G with Σ-labeled transitions, in which the payoff (value) of a play generating a word w is $\mathcal{A}(w)$, we have that G and $G \times \mathcal{A}$ have the same value [Boker and Lehtinen(2021)].

Generally speaking, we have the following relations between the notions.

— Determinisability by pruning $\Rightarrow^1$ history determinism $\Rightarrow^2$ good-for-gameness $\equiv^3$ good-for-automataness.
  (1) Determinisability by pruning is a special case of history-determinism, restricting the resolver to be positional in the automaton states.
  (2) A resolver for an automaton $\mathcal{A}$ can be combined with an optimal strategy in a game G, giving an optimal strategy in the game $G \times \mathcal{A}$ [Henzinger and Piterman(2006); Boker and Lehtinen(2021)].
  (3) A game is a special case of an alternating automaton $\mathcal{B}$, giving the implication from right to left, while for every specific word w, the value of $\mathcal{B}$ on w is viewed as a game, giving the implication from left to right [Boker and Lehtinen(2019), Theorem 16].
— History determinism $\Rightarrow$ good-for-MDPness. A resolver for $\mathcal{A}$ can be combined with an optimal strategy for an MDP M, ensuring an optimal strategy in $M \times \mathcal{A}$.
— Threshold-history-determinism $\equiv$ good-for-gameness for classes of automata for which the threshold letter-game is determined [Boker and Lehtinen(2021)].
— Good-for-gameness $\not\Rightarrow$ history determinism [Boker and Lehtinen(2021)].

While history-determinism and good-for-gameness do coincide for (ω-)regular automata [Boker and Lehtinen(2019)], this is not the case for quantitative automata. The gist of this separation is that in the composition with a quantitative game, Eve knows the value of the game, which gives her information on which value she needs to aim for when resolving the nondeterminism of the automaton, as in the case of threshold-history-determinism. Hence her strategy to resolve the nondeterminism of the automaton might depend on the game it is composed with. In contrast, a resolver for history-determinism is expected to build an optimal run on all words, without prior knowledge about their value. (See Fig. 3.)

Finally, in the ω-regular setting, good-for-gameness with respect to finite, infinite and infinitely branching games are all equivalent. However, this is not necessarily the case in general. For example, a pushdown automaton for the language a^*b that first pushes an arbitrary number n of a's onto the stack with ε-transitions and then reads a at most n times followed by a b is obviously not history-deterministic, yet it is good for finitely branching games: indeed, in a finitely branching game with winning condition a^*b, a winning strategy imposes a bound on the number of a's that any play that agrees with it can see; this bound can be used to push a sufficiently large number of a's onto the automaton's stack. Then, history-determinism for pushdown automata on finite words coincides with good-for-*infinitely-branching*-gameness, rather than good-for-*finitely-branching*-gameness [Guha et al.(2022), Section 8]. For pushdown automata on infinite words, the equivalence of history-determinism and good-for-gameness is open.

4. EXPRESSIVENESS

In the ω-regular setting, the appeal of nondeterministic automata lies mostly in their relative succinctness, when compared to deterministic ones, since all nondeterministic (and alternating) ω-regular automata are determinisable to, for example, parity automata. However, beyond the (ω-)regular setting, nondeterminism can also increase the expressiveness of automata models. This is the case, for example, for pushdown automata, Parikh automata, timed automata, register automata and various quantitative automata. This raises the question of the expressive power of history-determinism in these automata models.

In this section we present two ideas which can be used in various settings to show that history-deterministic automata are more expressive than deterministic ones. The first works already for finite words, but so far has only been applied to pushdown and

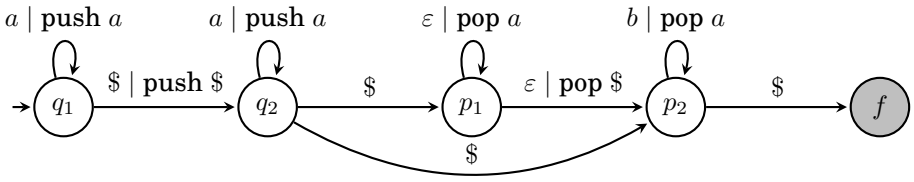

Fig. 4. A history-deterministic pushdown automaton recognising $\{a^n\$a^m\$b^k\$ \mid k \leq n \vee k \leq m\}$ from [Guha et al.(2021)]. The state f is final.

Parikh automata. The second is based on the coBüchi acceptance condition, and hence only makes sense for automata on infinite words, but can be adapted to many different automata models.

4.1. Automata on finite words

For regular automata on finite words, history-deterministic automata are not interesting in terms of expressiveness or succinctness, since they are all determinisable by pruning [Kupferman et al.(2006); Morgenstern(2003)]. However, when we consider automata with unbounded memory, such as pushdown automata, or with a more structured acceptance condition, such as Parikh automata, history-determinism can become more expressive than determinism. Indeed, a deterministic pushdown or Parikh automaton can only take part of the configuration into account to enable transitions: for pushdown automata only the state and top stack symbol can affect the available transitions, while for Parikh automata, only the current state matters. However, a resolver can use the whole configuration, including the stack contents or counter values, to resolve nondeterminism. We now exemplify how history-determinism can exploit this to be more expressive than determinism.

The language used to separate deterministic and history-deterministic pushdown automata is $\{a^n\$a^m\$b^k\$ \mid k \leq n \vee k \leq m\}$ [Guha et al.(2021)], consisting of two consecutive blocks of a's, of some lengths n and m, followed by a block of b's shorter than either m or n. Intuitively, this language is not recognised by a deterministic pushdown automaton since the hypothetical automaton would need to choose which block of a's to compare to the block of b's. However, the nondeterministic pushdown automaton in Fig 4 recognises this language. It pushes both a-blocks onto the stack, then nondeterministically chooses whether to compare the b-block to the last a-block (at the top of the stack), or to the first a-block (at the bottom of the stack).

This automaton is history-deterministic. Indeed, the only nondeterministic choice, which occurs before reading the first b, does not depend on the length of the b-block (*i.e.*, the future of the word) since the resolver can always compare the b-block to the longer of the a-blocks.

This automaton uses the stack to postpone the nondeterministic choice to the point at which the resolver has all the required information for it. In contrast, a nondeterministic automaton for the same language that only stores one (nondeterministically chosen) of the a-blocks would not be history-deterministic. This way of using history-determinism is not easy to adapt to other models, even to one-counter automata, which can be seen as pushdown automata with a single stack symbol.

Open: Are history-deterministic one-counter automata on finite words more expressive than deterministic ones?

Parikh automata, instead of adding a stack or counter to the automaton memory, add a set of counters to the acceptance condition. More precisely, the automaton can update a fixed set of counters along its run, however, the counter configuration is not used to enable or dissable transitions; at the end of the run, the automaton accepts if the counter configuration is in the semilinear set defining the automaton's accepting configurations. These recognise mildly quantitative languages, such as "is there a prefix with more a's than b's". This same language separates the classes of languages recognised by deterministic and history-deterministic Parikh automata [Erlich et al.(2022)]. Indeed, as with pushdown automata, the resolver has access to the whole configuration, which allows it to guess the right prefix, while a deterministic automaton can only access the counters to check for acceptance.

For other automata models, such as timed automata, no such argument can be made as history-deterministic timed automata on finite words are determinisable [Henzinger et al.(2022)]. As we will see next, a different proof strategy can be used on infinite words.

4.2. Automata on infinite words

On infinite words, the coBüchi acceptance condition seems particularly suitable for exploiting history-determinism. The core idea here is that the automaton must guess and check some infinitary behaviour. Using the coBüchi condition, the automaton can improve the guess an unbounded but finite number of times. Then, if the infinitary behaviour is such that there is a sequence of guesses that always converges to the correct one, the guessing can be done in a history-deterministic manner. We illustrate this somewhat abstract idea with a couple of concrete examples.

The following construction on one-counter coBüchi automata is inspired by Kuperberg and Skrzypczak's argument on the succinctness of history-deterministic coBüchi automata, described in Section 5.1. We consider an alphabet of pairs of increment ($+$), decrement ($-$) and no-op (0) symbols: $\left\{ \binom{+}{+}, \binom{+}{-}, \binom{-}{+}, \binom{-}{-}, \binom{+}{0}, \binom{0}{+}, \dots \right\}$. Words over this alphabet consist of words of pairs that can be viewed as pairs of words over $\{+, -, 0\}$. Every infix of each component word has an integer value, obtained by taking the difference between the number of $+$'s and the number of $-$'s. We say that a word over $\{+, -, 0\}$ is *safe* if it has no prefix with negative value. A word is *eventually safe* if it has a safe suffix, or, equivalently, if there is a lower bound to the value of its prefixes. Then, the language we are interested in, from [Lehtinen and Zimmermann(2022)], is the language of pairs of infinite words out of which at least one is eventually safe.

This language is not recognisable by a deterministic parity ω-one-counter (or pushdown) automaton as the hypothetical automaton would need to choose whether to track the value of the first or the second component. (The proof of this statement is somewhat trickier than the intuition would suggest, and we refer the reader to [Lehtinen and Zimmermann(2022)] for details.)

It is recognised by the nondeterministic coBüchi one-counter automaton in Fig 5. This automaton guesses nondeterministically (at any point) which component to track, and, whenever the tracked value goes below 0, sees a rejecting transition. It recognises the right language as whenever there is an eventually safe component, the accepting run can start tracking the right component at the start of the safe suffix. It is history-deterministic: the resolver can pick the component that has so far the longest suffix that could be extended into an infinite safe suffix; then, if one of the components is eventually safe, the resolver will eventually pick the right component to track at a safe suffix, and thus construct an accepting run (see [Lehtinen and Zimmermann(2022)] for details).

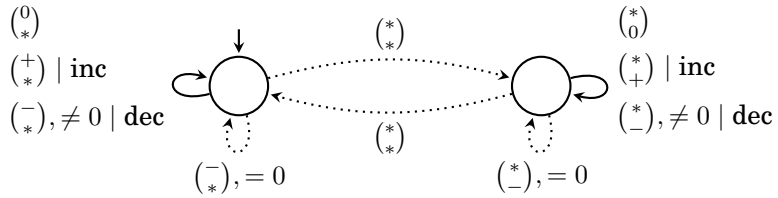

Fig. 5. A history-deterministic coBüchi one-counter automaton. The left state updates the counter according to the first component; the right according to the second. Dotted transitions are rejecting. A "$*$" value stands for all possible values.

A similar flavour of argument can be made for timed automata on infinite words, by considering the language of timed ω-words in which an event eventually occurs at each unit distance. The history-deterministic automaton for this language, in Fig. 6, must guess which unit interval to track, and when it makes a mistake, it is punished by a rejecting transition. The argument that it is indeed history-deterministic is similar to the above: the resolver can choose the time stamp which has the longest streak of unit-interval events so far, which guarantees that for any word in the language, the resolver eventually chooses to track an infinite series of events at unit interval. Again, the technically trickier part is to show that no deterministic timed automaton can recognise this language; we refer the reader to [Bose et al.(2022)].

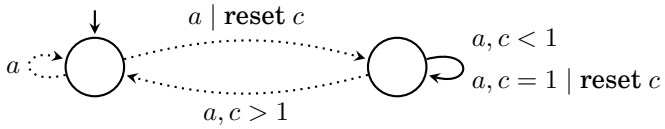

Fig. 6. A history-deterministic timed coBüchi automaton that accepts words in which there is a time t such that a occurs at $t + n$ for all integer n. The dotted transitions are rejecting.

Finally, a similar argument can be made in the context of register automata, for the language of blocks of data such that some data-value eventually occurs in every block.

Despite the apparent flexibility of this argument, we note that it relies on the coBüchi acceptance condition, and does not shed any light on, for example, the expressiveness gap between history-determinism and determinism in automata models that are based on the Büchi acceptance condition.

While history-determinism can be more powerful than determinism, it is usually not as powerful as full nondeterminism. Indeed, it is generally easy enough to imagine languages that require the automaton to guess the future of the word, for instance by implementing a dependence on the last, or last before something, letter. For example, the language $\{a^i\$a^j\$b^k x \mid (x = 0 \wedge k \leq i) \vee (x = 1 \wedge k \leq j)\}$ is not recognisable by a history-deterministic pushdown automaton as the nondeterminism depends on the last letter, but must be resolved earlier.

4.3. Closure properties

Closure properties of history-deterministic automata, whether it is closure under Boolean operations such as union, intersection and complementation in the Boolean setting, or closure under algebraic operations such as minimum, addition and subtraction in the quantitative setting, largely depend on the model in question. Unlike fully nondeterministic automata, for which union can easily be implemented by

adding a nondeterministic choice, history-deterministic automata are not necessarily closed under union. However, for some automata models, such as Büchi automata and timed automata, a product construction, similar to the one used for intersections, can work. In contrast, this fails for pushdown automata, where the product of two automata requires two stacks, and is therefore no longer a pushdown automaton. History-deterministic visibly pushdown automata, on the other hand, are closed under union and intersection without exponential blow-up because the stacks of the two automata can be unified into one.

5. SUCCINCTNESS

One of the appeals of history-deterministic automata is that they can, for some languages, be remarkably more succinct than any equivalent deterministic automaton. In this section we give an overview of known results on the succinctness of different history-deterministic models, when compared to deterministic ones and when compared to nondeterministic ones. We also discuss what happens when alternation is added to the mix.

5.1. Between determinism and history-determinism

The coBüchi case. The first result exhibiting the succinctness of history-deterministic automata, and, arguably, one of the seminal results demonstrating the relevance of history-determinism, is due to Kuperberg and Skrzypczak [Kuperberg and Skrzypczak(2015)]. The language they propose operates over an alphabet of partial permutations over $[n]$ (encoded into one of constant size 3); a sequence of letters can be seen as a braid of n parallel threads, some of which might occasionally be broken, as illustrated in Fig. 7. Then, the languages L_n of interest consist of the braids of width n that contain an infinite thread.

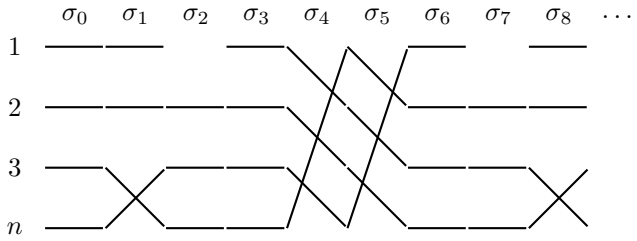

Fig. 7. A prefix of a word in Kuperberg's and Skrzypczak's language, represented as a braid of n threads. Each letter is a partial permutation, corresponding to a segment of the braid. A word is in the language if there is an infinite thread.

Kuperberg and Skrzypczak showed that any deterministic Rabin automaton recognising this language must be of size exponential in n. However, a little bit of nondeterminism lets us avoid this blow-up. Indeed, this language is easy enough to recognise with the nondeterministic coBüchi automaton $\mathcal{C}$ of size linear in n that nondeterministically chooses a thread to follow, and then uses the coBüchi condition to check that it is eventually infinite (see Fig. 8). This automaton, as the astute reader may suspect, is history-deterministic. Indeed, a resolver must guess which thread to follow, and eventually find an infinite thread, if it exists. One strategy to achieve this is to always choose the longest uninterrupted thread so far; another one is to cycle through the threads. In either case, whenever a word is in the language and there is an infinite thread, the resolver will eventually choose it and thus build an accepting run of $\mathcal{C}$.

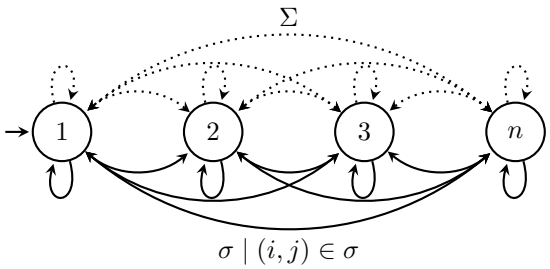

Fig. 8. A history-deterministic coBüchi automaton that is exponentially more succinct than any equivalent deterministic one [Kuperberg and Skrzypczak(2015)]. There is a non-rejecting transition between nodes i and j for a letter σ if (i, j) is in the permutation represented by σ. These transitions allow the automaton to track an uninterupted thread in a word. The dotted transitions (over all letters) are rejecting, and allow the automaton to change which thread it is tracking.

Note that this argument crucially relies on the coBüchiness of the automaton. No similar example is known to demonstrate the (at most quadratic [Kuperberg and Skrzypczak(2015), Theorem 8]) succinctness of history-deterministic Büchi automata, the existence of which remains an open problem.

Open: Is there a history-deterministic Büchi automaton that is (up to quadratically) smaller than any equivalent deterministic automaton?

For a while, this coBüchi argument seemed to be the only way in which history-determinism could be used to achieve succinctness. However, moving away from the regular setting to context-free languages opens up new ways to exploit the little nondeterminism allowed by history-determinism. Indeed, even on finite words, history-deterministic pushdown automata can be at least exponentially more succinct than any equivalent deterministic automaton [Guha et al.(2021)].

The languages in question are the fairly typical languages of bad n-bit counters: let c_n be the good n-bit counter, for instance $c_3 = 000\$001\$010 \dots \$111\$$; the family of languages L_n is then $L_n = \{w \in (\Sigma^n\$)^* \mid w \neq c_n\}$.

Any deterministic automaton recognising L_n, whether pushdown or not, must be of exponential size in n as it must, one way or another, remember n-bit counter values, either in the state-space or in the stack alphabet, to then check that a counting error occurs. However, a history-deterministic automaton can use the stack to store this information instead: it can push the input onto the stack, until a counting-error occurs. Then, the automaton nondeterministically identifies the counting-error and checks it with a small finite-state automaton while popping the stack. The nondeterministic choice consists of identifying when an error *has* occured and hence does not depend on the future of the word. In this manner, the stack can be used to implement a limited *two-way* type behaviour.

This example can be adapted to show that history-deterministic visibly pushdown automata can also be exponentially more succinct than deterministic ones [Guha et al.(2021)]. Since visibly pushdown automata allow an exponential determinisation procedure, this result is tight.

5.2. Between history-determinism and nondeterminism

The other side of the equation is the succinctness gap between fully nondeterministic automata and history-deterministic ones.

For ω-regular automata, it is clear that there is an exponential gap: The classic language that shows the exponential gap between deterministic and nondeterministic regular automata, which consists of finite words over $\{0,1\}$ in which the n^{th} bit from the end is a 1, can easily be adapted to infinite words, say by considering the n^{th} bit from the first $\#$ sign. As the language is recognised by a weak automaton, we get from [Boker et al.(2017), Corollary 18] that every history-deterministic Rabin or Street automaton for the language is determinisable-by-pruning, and thus exponentially bigger than the smallest nondeterministic automaton for the language.

In the case of pushdown automata this succinctness gap is particularly interesting since nondeterministic pushdown automata can be arbitrarily more succinct than deterministic ones [Valiant(1976); Hartmanis(1980)]. This means that either nondeterministic pushdown automata can also be arbitrarily more succinct than history-deterministic pushdown automata, or history-deterministic automata can be arbitrarily more succinct than deterministic ones. However, so far, the only lower bound on the succinctness gap between history-deterministic and nondeterministic pushdown automata is double-exponential [Guha et al.(2021)].

The language for this lower bound is again the above classic one, consisting of words in which the n^{th} bit from the end is a 1. Intuitively, a history-deterministic automaton cannot predict when the word ends, so it must remember the n last letters. However, a pushdown automaton can guess the n^{th} bit from the end and count to n using the stack and $\log n$ states (see [Guha et al.(2021), Theorem 5] for details).

Open: Can nondeterministic pushdown automata be arbitrarily more succinct than history-deterministic ones or can history-deterministic pushdown automata be arbitrarily more succinct than deterministic ones?

5.3. Alternating automata

Kuperberg and Skrzypczak's proof of succinctness for coBüchi history-deterministic automata (Section 5.1) can, without too much trouble, be extended to show that history-deterministic alternating automata are exponentially more succinct than nondeterministic *and* universal automata; For history-deterministic alternating parity automata, which enjoy an exponential determinisation procedure, this bound is tight [Boker et al.(2020b)].

We have mentioned that an alternating automaton can also be half-history-deterministic. It is open whether this intermediate class of automata can be double-exponentially more succinct than deterministic automata, already in the parity case.

Open: Is there a history-deterministic alternating Rabin or Street automaton or a half-history-deterministic alternating coBüchi, parity, Rabin, or Street automaton that is double-exponentially more succinct than any equivalent deterministic automaton of the same type?

6. DECIDING HISTORY-DETERMINISM

In this section we turn to the issue of determining whether an automaton is history-deterministic and discuss some of the tools and key arguments used in this line of research. Recall that deciding history-determinism corresponds to deciding the winner of the letter game that we used to define history-determinism: Adam builds, letter by letter, a word w, and Eve must build, transition by transition, an accepting run over w, in case that w is in the automaton's language.

Hardness. As sketched in Fig. 9, deciding history-determinism of a T-automaton is at least as hard as deciding the inclusion of a nondeterministic T automaton in a deterministic one [Lehtinen and Zimmermann(2022), Theorem 6.1], or solving T-games of the same type [Kuperberg and Skrzypczak(2015), Theorem 9]. In particular, deciding history-determinism for pushdown automata is undecidable, and for parity automata it is no easier than solving parity games with the same number of priorities.

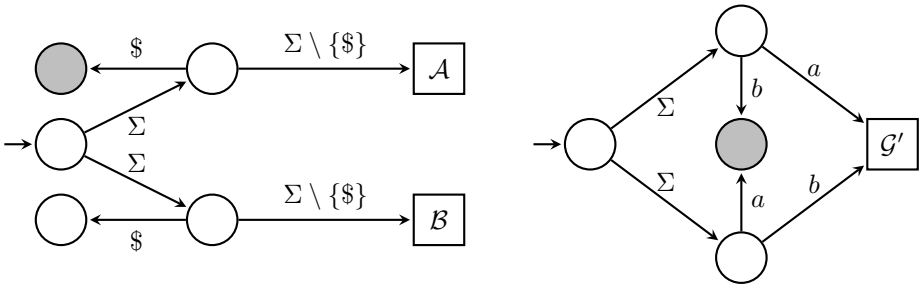

Fig. 9. Constructions demonstrating the hardness of deciding history-determinism; grey states are accepting sinks. Left: an automaton that is history-deterministic if and only if the language of the nondeterministic automaton $\mathcal{B}$ is included in the language of the deterministic automaton $\mathcal{A}$. Right: an automaton that is history-deterministic if and only if Player I wins the game $\mathcal{G}$, mimicked by the automaton $\mathcal{G}'$, in which moves of Player II are represented by letters and of Player I by choice of transitions.

The naive approach. If $\mathcal{A}$ is an ω-regular automaton, say a parity automaton, the naive approach to solving the letter game, and therefore deciding history-determinism, is to build an automaton that recognises its winning condition, which is ω-regular, and solve it as an ω-regular game. However, the resulting decision procedure is in EXPTIME. Equivalently, Henzinger and Piterman decide the history-determinism of $\mathcal{A}$ in EXPTIME by checking whether $\mathcal{A}$ simulates an equivalent deterministic automaton [Henzinger and Piterman(2006)]. This approach, in addition to being exponential, only works if determinisation is possible, which is not always the case, for example for timed automata, pushdown automata and various quantitative automata.

6.1. Token games

The *one-token-game* was introduced by Löding in an algorithm for deciding determinisability-by-pruning of a regular automaton, and was formally defined in [Löding and Repke(2013), Definition 5] in the course of extending this algorithm into a PTIME solution to the question of whether a regular automaton has an ℓ-lookahead delegator, for a given $\ell \in \mathbb{N}$. The game was later generalised by Bagnol and Kuperberg [Bagnol and Kuperberg(2018)] to a *k-token-game*, for a given $k \in \mathbb{N}$, in the course of seeking an easier to decide characterisation of history-determinism.

The *k-token-game* resembles the letter game, except that instead of just building a word, Adam also builds one or several (k to be precise) runs over the word, transition by transition, by moving k tokens along transitions of the automaton. Crucially, in the winning condition, the dependence on language membership is replaced by a condition on the runs built by Adam.

More precisely the k-token game G_k over $\mathcal{A}$ proceeds as follow. At each turn i:

— Adam chooses a letter $a_i \in \Sigma$;
— Eve responds with a transition τ_i over a_i;
— Adam chooses k transitions $\tau_{1,i} \dots \tau_{k,i}$ over a_i.

In the limit, a play consists of the word $w = a_0 a_1 \dots$ built by Adam, the run $\rho = \tau_0 \tau_1 \dots$ built by Eve and the k runs $\rho_j = \tau_{j,0} \tau_{j,1} \dots, j \in [1, k]$ built by Adam. Eve wins if either none of Adam's runs are accepting runs over w or if her run is an accepting run over w. In the quantitative setting, she wins if the value of her run is at least as high as the maximal value among Adam's runs. When clear from context, we overload notation by writing $G_k(A)$ both for the k-letter game over $\mathcal{A}$ and as a shorthand for "Eve wins $G_k(\mathcal{A})$".

Then, if one can show, for some class of automata, equivalence between the letter-game and some k-token-game, as well as an efficient solution to the k-token-game, we get an efficient procedure to decide history-determinism for this automata class. Due to the following lemma by Bagnol and Kuperberg, we need not look beyond $k = 2$:

> **Key Lemma:** For every automaton $\mathcal{A}$, if Eve wins $G_2(\mathcal{A})$ then she wins $G_k(\mathcal{A})$ for all k.

Bagnol and Kuperberg originally showed that this holds for ω-regular automata [Bagnol and Kuperberg(2018), Theorem 14], but it is in fact much more general, and holds for any common type of automata [Boker and Lehtinen(2022), Theorem 5.1]. The proof of this crucial lemma is rather clever. It proceeds by induction on the number of tokens Adam has and argues that if Eve has a winning strategy σ_k against Adam moving k tokens, and a winning strategy σ_2 against Adam moving 2 tokens, she can build a strategy to win against $k + 1$ tokens. To do so, she uses σ_k to play in $G_k(\mathcal{A})$ against Adam's first k tokens, and records her own moves with an additional *virtual token*. Then, to move in $G_{k+1}(\mathcal{A})$, she uses $\sigma_2(\mathcal{A})$, imagining that Adam plays with the virtual token and the last of Adam's actual tokens. Then, σ_k guarantees that if one of Adam's first k runs is accepting, so is the run built by the virtual token. Finally, σ_2 guarantees that if either the last of Adam's tokens or the virtual token traces an accepting run, then Eve's run is also accepting.

As a result we know that whenever some G_k characterises history-determinism, G_2 will do the trick. However, for many automata types, even G_1 suffices, as detailed in Section 6.2 and summarised in [Boker and Lehtinen(2022), Table 1].

Interestingly, there is no automata model for which it is known that the 2-token game does *not* characterise history-determinism[2]. In particular, Bagnol and Kuperberg conjectured that the 2-token game characterises history-determinism for parity automata (and therefore for all ω-regular automata). Since the 2-token game on parity automata with a fixed number of priorities is solvable in PTIME, if the conjecture holds, it implies a PTIME algorithm for deciding history-determinism of parity automata, for each number of priorities. This conjecture remains open to this day.

> **Open:** Does G_2 characterise history-determinism for parity automata?
> For what automata models does G_2 *not* characterise history-determinism?

6.2. When one token is enough

Bagnol and Kuperberg used G_1 in [Bagnol and Kuperberg(2018)] only to demonstrate that it does not characterise history-determinism for Büchi or coBüchi automata with the example depicted in Fig. 10. However, a couple of years after having been so speedily dismissed, G_1 proved its worth in characterising history-determinism in contexts beyond ω-regular automata.

[2]We consider automata on ω-words; the story is different (and out of our scope) for more exotic models.

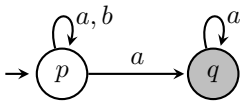

Fig. 10. An automaton for "eventually only a's", where q is an accepting sink. The automaton is not history-deterministic but Eve wins G_1 by waiting until Adam's token moves to q before moving her token to q.

First of all, G_1 characterises history-determinism on regular automata [Löding and Repke(2013)], as well as on all quantitative automata on finite words and those on infinite words having a safety acceptance condition [Boker and Lehtinen(2022)]. To see this, we argue as follows:

(1) A play for Eve in the letter game is losing if Eve chooses, after having built some run ρ over a word v, a transition τ such that there is a continuation vu in the language of the automaton without an accepting run with prefix $\rho\tau$. Indeed, when she makes such a move, Adam can win by playing u. We call such a move *non-cautious*, and a strategy *cautious* if it makes no non-cautious moves.
(2) A priori, unlike in the letter game, in G_1, non-cautious moves are not always fatal for Eve, as Adam's token might not be able to display an accepting run over the continuation u either. However, if Eve wins in G_1, she must, in particular, win against Adam's *copycat* strategies, in which Adam's token follows Eve's token until her first non-cautious move and then build an accepting run over the continuation that witnesses Eve's lack of caution. Such a play would be winning for Adam, so a winning strategy for Eve must avoid non-cautious moves against copycat strategies. Then, a winning strategy in G_1 for Eve becomes a cautious strategy for Eve in the letter game, as Eve simply imagines that Adam's token is following hers. Hence, if Eve wins G_1, she has a cautious strategy in the letter game.
(3) A cautious strategy in the letter game over a safety or finite word automaton is winning for Eve. Indeed, in both cases, a losing play for Eve must contain a non-cautious move [Boker and Lehtinen(2021)].

The same argument adapted to the quantitative setting shows that G_1 also characterises history-determinism for all quantitative automata on finite words.

Point (3) is straightforward for safety automata and automata on finite words, and also holds for Inf and DSum automata on infinite words, for which G_1 also characterises history-determinism [Boker and Lehtinen(2022)]. However, point (3) does not hold for all automata: for example, in the letter game over Reachability automata (on infinite words), a cautious play can be losing for Eve because she never reaches the target, while remaining in a region from where the target is reachable. (It turns out that G_1 nevertheless characterises history-determinism for Reachability automata, yet by a slightly more involved argument [Boker and Lehtinen(2022)].)

Due to this example, whether cautious strategies are winning in the letter game is not a necessary condition for G_1 to characterise history-determinism.

Open: For which value functions Val, does G_1 characterise history-determinism of Val-automata on infinite words?

Overall, compared to its modest role in [Bagnol and Kuperberg(2018)], the one-token game has turned out to be a surprisingly powerful tool, especially in the quantitative setting. However, it has its limit, and, as shown in the example of Fig. 11, it does not characterise history-determinism for Sup automata on infinite words. This is perhaps slightly surprising, given the duality between the Inf and Sup payoff functions.

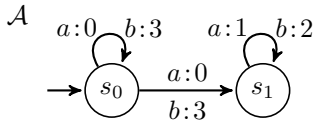

Fig. 11. A Sup automaton $\mathcal{A}$, demonstrating that G_1 does not characterize history-determinism for Sup automata on infinite words. $\mathcal{A}$ is not history-deterministic [Boker and Lehtinen(2022)]: Adam can play a when Eve's run is in s_0 and b when Eve's run is in s_1. If Eve stays in s_0, then the word has value 1 and Eve's run has value 0; if Eve goes to s_1, then the word has value 3 but Eve's run has value 2. Eve wins G_1 by moving to s_1 once Adam's token is in s_1. If Adam stays in s_0, they have the same run; if Adam moves and plays b before Eve moves, she gets value 3 and wins; if he doesn't, then Eve gets the same value as Adam.

6.3. Two tokens

We now move onto discussing when the 2-token game is known to characterise history-determinism. To do so, we will use an additional game, which we denote by $\mathsf{HD}_k(\mathcal{A})$, that generalises the letter game, which corresponds to $\mathsf{HD}_1(\mathcal{A})$. In $\mathsf{HD}_k(\mathcal{A})$, as in the letter game, Adam builds a word w letter by letter, but this time Eve constructs k runs, rather than just one, transition by transition. She wins if w is not in $\mathcal{A}$'s language or if any of her runs are accepting. For a quantitative automaton, at least one of her runs must match the value of the automaton on w.

More precisely, in the game HD_k over an automaton $\mathcal{A}$, at each turn i:

— Adam chooses a letter $a_i \in \Sigma$;
— Eve responds with k transitions $\tau_{1,i} \dots \tau_{k,i}$ over a_i.

In the limit, a play consists of the word $w = a_0 a_1 \dots$ built by Adam and the k runs $\rho_j = \tau_{j,0}\tau_{j,1} \dots, j \in [1, k]$ built by Eve. It is winning for Eve if either w is not in the language of $\mathcal{A}$ or if ρ_j is an accepting run over w for some $j \in [1, k]$. In the quantitative setting, Eve wins if the value of some ρ_j matches the value of the automaton on w.

As before, we use $\mathsf{HD}_k(\mathcal{A})$ both to mean the game HD_k over $\mathcal{A}$ and, when it is clear from context, as a shorthand for "Eve wins $\mathsf{HD}_k(\mathcal{A})$".

By definition, Eve wins $\mathsf{HD}_1(\mathcal{A})$ exactly when $\mathcal{A}$ is history-deterministic. Furthermore, each HD_k characterises a class of automata between full nondeterminism and history-determinism and can be thought of as a measure of nondeterminism, similar to notions of bounded ambiguity (see [Boker(2022)] for more on such measures). That being said, while the hierarchy of nondeterminism induced by HD_k is interesting in its own right [Hazard and Kuperberg(2023)], here we use HD_k as a technical tool.

6.3.1. G_2 for Büchi and coBüchi automata. We sketch the argument used to show that G_2 charcterises history-determinism for Büchi [Bagnol and Kuperberg(2018)] and coBüchi [Boker et al.(2020a)] automata. In both cases, we assume, towards contradiction, that Eve wins $G_2(\mathcal{A})$ on an automaton $\mathcal{A}$ that is not history-deterministic. The argument then proceeds as follows:

— The determinacy and regularity of the letter-game on Büchi and coBüchi automata implies the existence of a *finite memory* winning strategy τ for Adam in $\mathsf{HD}_1(\mathcal{A})$. This strategy produces letters and builds words in $L(\mathcal{A})$.
— We then build a strategy to move a large number k of tokens against τ in $\mathsf{HD}_k(\mathcal{A})$. More precisely, picking a large k that depends on the memory size of τ, we build a strategy σ for Eve in $\mathsf{HD}_k(\mathcal{A})$, which is not necessarily winning in general, but guarantees that if Adam plays any word produced by τ, one of Eve's k tokens traces an accepting run.
— Then, we recall the key lemma, according to which, since Eve wins $G_2(\mathcal{A})$, she also wins $G_k(\mathcal{A})$ with some strategy σ'. To conclude the argument, we build a strategy

for Eve that is not winning in general, but wins against τ: Eve imagines k virtual tokens, which she moves according to σ, and chooses her moves in $\mathsf{HD}_1(\mathcal{A})$ as if she was using σ' in $G_k(\mathcal{A})$ against the k virtual tokens. The result is a strategy that wins against τ: indeed, σ guarantees that one of the virtual tokens traces an accepting run and σ' therefore guarantees that Eve's run in $\mathsf{HD}_1(\mathcal{A})$ is accepting.

The construction of the strategy σ, which moves a large number of tokens against τ, is rather different for Büchi and coBüchi automata, as it uses in each case the specificities of the acceptance condition. This makes it difficult to generalise it to the parity condition, even with just 3 priorities. Further, note that this argument is not constructive: while it argues that Eve wins $\mathsf{HD}(\mathcal{A})$ whenever she wins $G_2(\mathcal{A})$, it does not detail how to build a winning strategy. A more direct proof that would turn a strategy in G_2 into a strategy in $\mathsf{HD}(\mathcal{A})$ might bring some helpful insight that could potentially also help understand the parity case.

Open: How can a winning strategy for Eve in G_2 on a Büchi or coBüchi automaton $\mathcal{A}$ be turned into a winning strategy for Eve in $\mathsf{HD}(\mathcal{A})$, *i.e.,* into a resolver for $\mathcal{A}$?

Note that Boker *et al.* further showed that if G_2 characterises history-determinism for parity automata, then an alternating version of G_2, also solvable in PTIME, characterises history-determinism for alternating parity automata [Boker et al.(2020a), Section 5]. In contrast, deciding *half history-determinism*, that is whether the nondeterminism of an alternating automaton is history-deterministic, is PSPACE-complete already on finite words [Boker et al.(2020a), Section 3.2].

6.3.2. G_2 for quantitative automata. Beyond ω-regular automata, the same tools can be used to show that G_2 characterises history-determinism for some quantitative automata, and in particular LimSup, LimInf and Sup [Boker and Lehtinen(2021)].

In the LimSup and LimInf cases, we do not need to rely on a finite memory strategy for Adam in the letter game (in fact, it isn't obvious that such a strategy must exist), nor even determinacy. Instead, we use the three following observations that we have already made:

(1) $\mathsf{HD}(\mathcal{A}) \implies G_k(\mathcal{A})$
If Eve wins the letter-game on an automaton $\mathcal{A}$, then she wins any k-token-game on $\mathcal{A}$, by ignoring Adam's runs.
(2) $G_2(\mathcal{A}) \implies \forall k.G_k(\mathcal{A})$
If Eve wins $G_2(\mathcal{A})$, then she wins $G_k(\mathcal{A})$ for all k (the key lemma).
(3) $(\exists k.G_k(\mathcal{A}) \wedge \mathsf{HD}_k(\mathcal{A})) \implies \mathsf{HD}(\mathcal{A})$
If Eve wins both $G_k(\mathcal{A})$ and $\mathsf{HD}_k(\mathcal{A})$ for some k, then she also wins $\mathsf{HD}(\mathcal{A})$, that is, $\mathcal{A}$ is history-deterministic. This is implicit in the Büchi and coBüchi arguments: Eve can build a strategy in $\mathsf{HD}(\mathcal{A})$ by imagining k virtual tokens that move according to her strategy in $\mathsf{HD}_k(\mathcal{A})$, and choose her moves in $\mathsf{HD}(\mathcal{A})$ using a winning strategy in $G_k(\mathcal{A})$ against the virtual tokens.

While these alone do not suffice, if one can show that for $\mathcal{A}$ in a given class of automata, $G_2(\mathcal{A}) \implies \exists k.\mathsf{HD}_k(\mathcal{A})$, (that is, whenever Eve wins G_2 on $\mathcal{A}$, there is some k such that she can find an accepting run with k tokens), one can conclude that G_2 characterises history-determinism. Indeed, if $G_2(\mathcal{A})$, then, by assumption $\mathsf{HD}_k(\mathcal{A})$ for some k and from (2), we also have $G_k(\mathcal{A})$. Then, from (3), we conclude $\mathsf{HD}(\mathcal{A})$. The other direction is given by (1).

This means that one strategy to show that history-determinism is decidable via G_2 is to show that $G_2(\mathcal{A}) \implies \exists k.\mathsf{HD}_k(\mathcal{A})$. This is the case for LimSup, LimInf and Sup automata on infinite words [Boker and Lehtinen(2022)].

6.4. Challenges in more complex models

When we move beyond ω-regular automata, neither the determinacy of the letter-game, nor the existence of finite-memory strategies for either player, is a given, which adds a layer of difficulty to reasoning about history-determinism. Indeed, it is not clear for which acceptance conditions (beyond safety and reachability) the letter game for timed automata or register automata is determined, due, at least in part, to it not being obvious whether nondeterministic timed or register automata recognise languages that are Borel. Furthermore in these more complex models, strategies in the letter game are no longer finitely branching, either because Adam picks a letter from an infinite or timed alphabet (in the register and timed cases, respectively), or because Eve can choose to play arbitrarily many epsilon transitions (for example in the pushdown case). These factors make reasoning about history-determinism somewhat trickier for these models, and can cause proof strategies from the Boolean setting to fail.

7. APPLICATIONS

So far, we have discussed to what extent history-deterministic automata enjoy the expressiveness and succinctness of nondeterministic automata. The other side of the equation is that they enjoy some of the better algorithmic properties of determinism, which makes them appealing for applications such as model-checking and synthesis.

7.1. Model-checking

Language inclusion is one of the core algorithmic problems of automata-theoretic model-checking. Indeed, with both system and specification represented as automata, language inclusion captures whether all behaviours of the system are permitted by the specification.

For history-deterministic automata, language inclusion reduces to fair simulation, which is typically algorithmically easier to decide than inclusion. Fair simulation [Henzinger et al.(2002)] is defined by the simulation game, where a Spoiler chooses a transition in the automaton to be simulated, and Duplicator chooses a transition over the same letter in the simulating automaton. Duplicator wins if, in the limit, whenever Spoiler has built an accepting run, so has Duplicator. If an automaton $\mathcal{A}$ simulates an automaton $\mathcal{B}$ then the language of $\mathcal{B}$ is included in the language of $\mathcal{A}$. In general, there is no converse implication. However, it is easy to see that a history-deterministic automaton $\mathcal{A}$ simulates any automaton of which the language is included in the language of $\mathcal{A}$ by using the resolver strategy to choose transitions in the simulation game. Hence, for history-deterministic automata, language inclusion is no harder than fair simulation.

If we consider the more general setting of labelled transitions systems (LTSs), then history-deterministic LTSs are *exactly* those for which inclusion and fair simulation coincide [Henzinger et al.(2022), Theorem 4][3]. However, it is not clear for which classes of automata this also holds.

[3]This is also related to *guidability* [Colcombet and Löding(2008)].

Open: For which classes C of automata are the following equivalent for $\mathcal{A} \in C$:

(1) $\mathcal{A}$ is history-deterministic.
(2) For all $\mathcal{B} \in C$, it is the case that $L(\mathcal{B}) \subseteq L(\mathcal{A})$ if and only if $\mathcal{A}$ simulates $\mathcal{B}$.

Note that this is the case for automata that are determinisable: if language inclusion is equivalent to simulation, then a determinisable automaton has a resolver by simulating the equivalent deterministic automaton.

7.2. Church synthesis

One of the applications of history-deterministic automata, which has earned them the title of good-for-games, is the synthesis problem, modelled as a game. In Church synthesis [Church(1963)], the interaction of a system with its environment is represented by a system player set against an antagonistic environment. The players alternate picking moves, represented either as outgoing edges in an arena or as letters of an alphabet, thus building an infinite sequence of moves that represents the interaction of the system and its environment. The goal of the system player is to ensure that this sequence, seen as a word, is in a language called the *winning condition*, which represents the specification, while the goal of the adversarial environment is to prevent this. Solving the synthesis problem corresponds to deciding whether there is a winning strategy for the system player, and, when possible, finding it.

To solve a game in which the winning condition is given by a deterministic automaton, it suffices to take the product of the arena and the automaton, and solve the resulting game, of which the winning condition depends on the acceptance condition of the deterministic automaton. For instance, to solve a game with a finite arena and an ω-regular winning condition L, it suffices to take the product of the arena with a deterministic parity automaton for L and solve the resulting parity game. For ω-deterministic context-free winning conditions, we take the product with the deterministic parity pushdown automaton for the winning condition; the resulting game is a parity game on a pushdown arena.

This reduction fails for general nondeterministic automata. Intuitively, this is because the product construction yields a game in which nondeterministic choices have to be resolved before the full path in the arena is determined, and might therefore influence the strategy of the opponent. Yet, full determinism is not required for the reduction to work: history-determinism—or, more precisely, good-for-gameness—suffices. Hence solving games with winning conditions captured by history-deterministic automata is no harder than for deterministic automata of the same type.

7.3. Good-enough or best-value synthesis

In the Church synthesis problem, the system must guarantee that the specification is fulfilled *whatever* the environment does. This is often a stronger requirement than reasonable. For example, in the canonical example of the coffee machine, if the users (i.e., the environment) do not fill the water tank when it is empty, then the machine won't produce coffee, hence failing to satisfy its specification. Then, according to the Church synthesis problem, the specification is not realisable and we must give up coffee. This is due to the assumption that the environment is entirely antagonistic, and will always enact the worst-case scenario.

Instead, intent on getting our caffeine hit, we can argue that rather than requiring the system to satisfy the specification *come what may*, we could be content with a system that satisfies the specifications *whenever possible, given the environment*

behaviour. This is the idea of *good-enough synthesis*, as described by Almagor and Kupferman [Almagor and Kupferman(2020)].

In the quantitative setting, the simple synthesis problem, described by Bloem, Chatterjee, Henzinger and Jobstmann [Bloem et al.(2009)], has a similar issue: since we assume that the environment will always behave antagonistically and only take into account the worst-case outcome of the system's strategy, the system has no incentive to do better in other scenarios. In practice, this might mean that if the shortest route to the destination uses a bridge that is closed on days with high wind, a navigator system might always suggest a longer route that avoids the bridge, regardless of whether the shorter route is available, since it will only care about the performance in the worst-case scenario. Instead, we would of course like the navigator to suggest the shorter route whenever the environment allows it. The corresponding version of the synthesis problem is *best-value synthesis* (as in [Filiot et al.(2020)]), which, instead of asking the system to guarantee the optimal value among all behaviours, asks the system to guarantee the optimal value, *given the environment behaviour*. The following two problems are polynomially equivalent:

(1) Deciding good-enough synthesis (or best-value synthesis in the quantitative case) of a specification given by a deterministic automaton, and
(2) Deciding whether a nondeterministic automaton of the same type is history-deterministic.

We sketch the proof of this equivalence. In the synthesis problem, the system and environment players construct a pair of words, one over an input alphabet Σ_I (controlled by the environment), and one over an output alphabet Σ_O (controlled by the system). Given a deterministic automaton $\mathcal{A}$ over $\Sigma_I \times \Sigma_O$ that describes a specification language for which we would like to decide the synthesis problem, we construct a nondeterministic automaton $\mathcal{A}'$ over Σ_I that recognises the projection of $L(\mathcal{A})$ on the first component. $\mathcal{A}'$ is similar to $\mathcal{A}$ except that it only operates on the input alphabet Σ_I and hence for each transition of $\mathcal{A}$ over a pair of letters $\binom{a}{b} \in \Sigma_I \times \Sigma_O$, $\mathcal{A}'$ has a transition only over the letter $a \in \Sigma_I$. $\mathcal{A}'$ is nondeterministic as it might have several transitions over the same input letter from each state, up to one for each letter of Σ_O.

It is easy to see that $\mathcal{A}'$ constructed this way recognises the projection of $L(\mathcal{A})$ on the first component. Furthermore, it is history-deterministic if and only if $L(\mathcal{A})$ is good-enough realisable: a resolver for $\mathcal{A}'$ corresponds exactly to a solution system for the good-enough synthesis problem as constructing an accepting run in $\mathcal{A}'$ induces a choice of output letters in $\mathcal{A}$.

For the other direction, to decide whether a nondeterministic automaton $\mathcal{A}$ is history-deterministic, it suffices to decide good-enough synthesis for the following deterministic automaton $\mathcal{A}'$ recognising pairs $\binom{w}{\rho}$, where ρ is an accepting run of $\mathcal{A}$ over w. It is constructed by turning a transition t over a letter a in $\mathcal{A}$ into a transition over $\binom{a}{t}$ in $\mathcal{A}'$. Then, a solution to the good-enough synthesis problem for $\mathcal{A}'$ corresponds exactly to a resolver for $\mathcal{A}$ as it constructs an accepting run for each word $w \in L(\mathcal{A})$ on the fly, transition by transition.

In the quantitative setting, by a similar argument, deciding history-determinism is polynomially equivalent to the best-value synthesis problem.

8. SIZE AND MODELS OF RESOLVERS

The definition of history-determinism stipulates the existence of a resolver, *i.e.*, a winning strategy in the letter game, without restrictions on its size, *i.e.*, the size of the memory used by the strategy, or, if infinite, on its structure (e.g., counter or stack(s)). In some situations, the size of the resolver doesn't matter, for instance, when solving

games: the complexity of solving a game with a winning condition given by a history-deterministic automaton, and the size of a winning strategy in such a game, does not depend on the size of the resolver. The size of the resolver can be seen, in some sense, as the gain in efficiency between a deterministic and history-deterministic automaton.

However, at times the size and structure of the resolver *does* matter. For instance, in the reduction between solving good-enough (or best-value) synthesis and deciding history-determinism, the resolver corresponds to the solution system. In this case, the implementability of the resolver is of great importance.

In this section we discuss when simple resolvers suffice, when they do not, and some of the open questions that remain. The simplest resolvers are positional, depending only on the positions of the letter game on the automaton $\mathcal{A}$. If $\mathcal{A}$ has no external memory, meaning that a position of the letter game is just a pair of $\mathcal{A}$'s state and an input letter, then $\mathcal{A}$ admits a positional resolver if and only if it is determinisable-by-pruning. In general, we represent resolvers as transducers. Finite memory strategies correspond to finite-state transducers, but we also consider unbounded state transducers, such as pushdown transducers.

Computing resolvers. Computing the resolver corresponds to computing a winning strategy in the letter-game. In several cases the problem of computing the resolver is more complex than just deciding the winner: for example, deciding whether a coBüchi automaton is history-deterministic is in PTIME, even though history-deterministic coBüchi automata can require exponential-sized resolvers [Boker et al.(2020a); Kuperberg and Skrzypczak(2015)].

8.1. Resolvers for (ω-)regular automata

History-deterministic regular automata are always determinisable-by-pruning [Kupferman et al.(2006)], thus admiting positional resolvers. In the ω-regular case, it depends on the acceptance condition. Safety, reachability, and weak automata are also history-deterministic if and only if they are determinisable-by-pruning [Boker et al.(2017), Theorem 17], so they have positional resolvers. History-deterministic Büchi automata can be determinised with up to a quadratic size blow-up [Kuperberg and Skrzypczak(2015)], implying polynomial resolvers.

For the other classical acceptance conditions, and in particular for coBüchi, parity, Street, Rabin, and Muller, the letter-game on an automaton $\mathcal{A}$ can be seen as a game with an ω-regular winning condition, captured by a deterministic parity automaton of size exponential in the size of $\mathcal{A}$. It can therefore be encoded into a parity game of exponential size in $\mathcal{A}$, which has a positional winning strategy. It follows that when $\mathcal{A}$ is history-deterministic it has a resolver of size exponential in the size of $\mathcal{A}$. This upper bound is matched by Kuperberg and Skrzypczak's lower bound on coBüchi automata: the resolver of the automaton in Section 5.1 must be of exponential size, as else it could be used to determinise the automaton without exponential blow-up.

Moving to ω-regular automata with stronger acceptance conditions, such as Emerson-Lei automata, their determinisation might involve a doubly-exponential size blow-up [Safra and Vardi(1989), Proposition 4.5 and Conclusions], implying a double-exponential upper bound on the size of a resolver.

8.2. Resolvers for pushdown automata

The history-deterministic pushdown automaton in Section 4 recognising the language $\{a^n\$a^m\$b^k\$ \mid k \leq n \vee k \leq m\}$ admits a pushdown resolver that uses the stack to compare the length of the first and second a-blocks. However, it is easy to extend this example to a history-deterministic pushdown automaton without a pushdown resolver. Indeed,

consider the similar language $\{a^n\$a^m\$a^\ell\$b^k\$ \mid k \leq n \vee k \leq m \vee k \leq \ell\}$, in which instead of just two a-blocks, there are three a-blocks. The b-block must still be shorter than one of the a-blocks. Then, the resolver must compare the length of *three* a-blocks, which is not something a pushdown machine can achieve [Guha et al.(2021)].

This shows that not all history-deterministic pushdown automata enjoy pushdown resolvers. In fact, it is open whether all history-deterministic pushdown automata even have a Turing-*computable* resolver.

Open: Do history-deterministic pushdown automata always admit Turing-computable resolvers?

On the other hand, history-deterministic visibly pushdown automata, which, as we have discussed, are better behaved than general pushdown automata, admit visibly pushdown resolvers. This is due to the letter-game reducing, in this setting, to a game on a visibly pushdown arena, which admits a visibly pushdown winning strategy.

8.3. Resolvers for quantitative automata

Quantitative automata may have many different value functions, and the size of resolvers varies accordingly. We list below some of the value functions for which the resolver size, or an upper bound for it, is known.

History-deterministic Sum and Avg automata on finite words and DSum on finite and infinite words are determinisable-by-pruning [Boker and Lehtinen(2021), Theorems 21 and 23] and [Hunter et al.(2016), Section 5], thus have positional resolvers.

Sup automata on finite words and Inf automata on both finite and infinite words admit polynomial resolvers, obtained by solving the corresponding one-token games, whose winning strategy in these cases can be adapted to a resolver of the same memory size [Boker and Lehtinen(2022)].

Sup, LimInf and LimSup automata on infinite words admit resolvers of up to a single exponential size [Boker and Lehtinen(2022)], where for LimInf automata it is also optimal, as they generalise coBüchi automata.

9. CONCLUSIONS

History-determinism has the potential to combine some of the best aspects of both deterministic and nondeterministic automata, having some algorithmic properties of the first and some of the expressiveness and succinctness of the second. Its dual allure comes both from its theoretical appeal as a natural restriction of nondeterminism (as witnessed by its many interesting properties and related definitions) and from its potential impact on practical applications such as synthesis. Beyond the material covered by this survey, it remains the subject of intense research efforts, both with respect to fundamental questions, such as whether history-determinism is always characterised by two-token games, known as the G_2 conjecture, and with respect to its peripheral extent, that is, its role in different automata models. Translating the theoretical potential of history-determinism into practical verification and synthesis tools remains largely uncharted territory.

ACKNOWLEDGMENTS

We thank the many colleagues who gave us feedback on a draft of this article. Research of Udi Boker supported by the Israel Science Foundation grant 2410/22.

REFERENCES

Bader Abu Radi and Orna Kupferman. 2022. Minimization and Canonization of GFG Transition-Based Automata. *Log. Methods Comput. Sci.* 18, 3 (2022). DOI:https://doi.org/10.46298/lmcs-18(3:16)2022

Shaull Almagor and Orna Kupferman. 2020. Good-Enough Synthesis. In *Proceedings of CAV (Lecture Notes in Computer Science, Vol. 12225)*. Springer, 541–563. DOI:https://doi.org/10.1007/978-3-030-53291-8_28

Benjamin Aminof, Orna Kupferman, and Robby Lampert. 2010. Reasoning about online algorithms with weighted automata. *ACM Trans. Algorithms* 6, 2 (2010), 28:1–28:36. DOI:https://doi.org/10.1145/1721837.1721844

Marc Bagnol and Denis Kuperberg. 2018. Büchi good-for-games automata are efficiently recognizable. In *Proceedings of FSTTCS*. 16. DOI:https://doi.org/10.4230/LIPIcs.FSTTCS.2018.16

Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. 2009. Better Quality in Synthesis through Quantitative Objectives. In *Proceedings of CAV (Lecture Notes in Computer Science, Vol. 5643)*, Ahmed Bouajjani and Oded Maler (Eds.). Springer, 140–156. DOI:https://doi.org/10.1007/978-3-642-02658-4_14

Udi Boker. 2018. Why These Automata Types?. In *Proceedings of LPAR*. 143–163. DOI:https://doi.org/10.29007/c3bj

Udi Boker. 2021. Quantitative vs. Weighted Automata. In *Proc. of Reachbility Problems*. 1–16. DOI:https://doi.org/10.1007/978-3-030-89716-1_1

Udi Boker. 2022. Between Deterministic and Nondeterministic Quantitative Automata. In *Proc. of CSL*. 1:1–1:15. DOI:https://doi.org/10.4230/LIPIcs.CSL.2022.1

Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. 2013. Nondeterminism in the Presence of a Diverse or Unknown Future. In *Proceedings of ICALP*. 89–100. DOI:https://doi.org/10.1007/978-3-642-39212-2_11

Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak. 2020a. On Succinctness and Recognisability of Alternating Good-for-Games Automata. *arXiv preprint* (2020). DOI:https://doi.org/10.48550/arXiv.2002.07278

Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak. 2020b. On the Succinctness of Alternating Parity Good-For-Games Automata. In *Proceedings of FSTTCS*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. DOI:https://doi.org/10.4230/LIPIcs.FSTTCS.2020.41

Udi Boker, Orna Kupferman, and Michał Skrzypczak. 2017. How Deterministic are Good-For-Games Automata?. In *Proceedings of FSTTCS*. 18:1–18:14. DOI:https://doi.org/10.4230/LIPIcs.FSTTCS.2017.18

Udi Boker and Karoliina Lehtinen. 2019. Good for Games Automata: From Nondeterminism to Alternation. In *Proceedings of CONCUR (LIPIcs, Vol. 140)*. 19:1–19:16. DOI:https://doi.org/10.4230/LIPIcs.CONCUR.2019.19

Udi Boker and Karoliina Lehtinen. 2021. History Determinism vs. Good for Gameness in Quantitative Automata. In *Proc. of FSTTCS*. 35:1–35:20. DOI:https://doi.org/10.4230/LIPIcs.FSTTCS.2021.38

Udi Boker and Karoliina Lehtinen. 2022. Token Games and History-Deterministic Quantitative Automata. In *FOSSACS*. 120–139. A submitted journal version is available at https://arxiv.org/abs/2110.14308.

Sougata Bose, Thomas A. Henzinger, Karoliina Lehtinen, Sven Schewe, and Patrick Totzke. 2022. History-Deterministic Timed Automata Are Not Determinizable. In *Proceedings of RP (Lecture Notes in Computer Science, Vol. 13608)*, Anthony W. Lin, Georg Zetzsche, and Igor Potapov (Eds.). Springer, 67–76. DOI:https://doi.org/10.1007/978-3-031-19135-0_5

Cristian S Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. 2017. Deciding parity games in quasipolynomial time. In *Proceedings of STOC*. 252–263. DOI:https://doi.org/10.1137/17M1145288

Antonio Casares. 2022. On the Minimisation of Transition-Based Rabin Automata and the Chromatic Memory Requirements of Muller Conditions. In *Proceedings of CSL (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 216)*, Florin Manea and Alex Simpson (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 12:1–12:17. DOI:https://doi.org/10.4230/LIPIcs.CSL.2022.12

Antonio Casares, Thomas Colcombet, and Karoliina Lehtinen. 2022. On the Size of Good-For-Games Rabin Automata and Its Link with the Memory in Muller Games. In *Proceedings of ICALP (LIPIcs, Vol. 229)*, Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 117:1–117:20. DOI:https://doi.org/10.4230/LIPIcs.ICALP.2022.117

Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. 2010. Quantitative languages. *ACM Trans. Comput. Log.* 11, 4 (2010), 23:1–23:38. DOI:https://doi.org/10.1145/1805950.1805953

Alonzo Church. 1963. Application of recursive arithmetic to the problem of circuit synthesis. *J. of Symbolic Logic* 28, 4 (1963). DOI:https://doi.org/10.2307/2271310

Thomas Colcombet. 2009. The theory of stabilisation monoids and regular cost functions. In *Proceedings of ICALP*. 139–150. DOI:https://doi.org/10.1007/978-3-642-02930-1_12

Thomas Colcombet. 2013. Fonctions régulières de coût. *Habilitation à diriger les recherches, École Doctorale de Sciences Mathématiques de Paris Centre* (2013). \url{https://www.irif.fr/~colcombe/Publications/habilitation-colcombet_v1.1.pdf}

Thomas Colcombet and Nathanaël Fijalkow. 2016. The Bridge Between Regular Cost Functions and Omega-Regular Languages. In *Proceedings of ICALP*. 126:1–126:13. DOI:https://doi.org/10.4230/LIPIcs.ICALP.2016.126

Thomas Colcombet and Christof Löding. 2008. The Non-deterministic Mostowski Hierarchy and Distance-Parity Automata. In *Proceedings of ICALP*, Vol. 5126. 398–409. DOI:https://doi.org/10.1007/978-3-540-70583-3_33

Thomas Colcombet and Cristof Löding. 2010. Regular Cost Functions over Finite Trees. In *Proceedings of LICS*. 70–79. DOI:https://doi.org/10.1109/LICS.2010.36

Enzo Erlich, Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. 2022. History-deterministic Parikh Automata. *CoRR* abs/2209.07745 (2022). DOI:https://doi.org/10.48550/arXiv.2209.07745 arXiv:2209.07745

Emmanuel Filiot, Ismaël Jecker, Nathan Lhote, Guillermo A. Pérez, and Jean-François Raskin. 2017. On delay and regret determinization of max-plus automata. In *LICS*. 1–12. DOI:https://doi.org/10.1109/LICS.2017.8005096

Emmanuel Filiot, Christof Löding, and Sarah Winter. 2020. Synthesis from Weighted Specifications with Partial Domains over Finite Words. In *Proceedings of FSTTCS (LIPIcs, Vol. 182)*, Nitin Saxena and Sunil Simon (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 46:1–46:16. DOI:https://doi.org/10.4230/LIPIcs.FSTTCS.2020.46

Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. 2021. A Bit of Nondeterminism Makes Pushdown Automata Expressive and Succinct. In *Proc. of MFCS*. 53:1–53:20. DOI:https://doi.org/10.4230/LIPIcs.MFCS.2021.53

Shibashis Guha, Ismal Jecker, Karoliina Lehtinen, and Martin Zimmermann. 2022. A Bit of Nondeterminism Makes Pushdown Automata Expressive and Succinct. DOI:https://doi.org/10.48550/ARXIV.2105.02611

Juris Hartmanis. 1980. On the Succinctness of Different Representations of Languages. *SIAM J. Comput.* 9, 1 (1980), 114–120. DOI:https://doi.org/10.1137/0209010

Émile Hazard and Denis Kuperberg. 2023. Explorable automata. In *Proc. of CSL*.

Thomas Henzinger and Nir Piterman. 2006. Solving games without determinization. In *Proceedings of CSL*. 395–410. DOI:https://doi.org/10.1007/11874683_26

Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. 2002. Fair Simulation. *Inf. Comput.* 173, 1 (2002), 64–81. DOI:https://doi.org/10.1006/inco.2001.3085

Thomas A. Henzinger, Karoliina Lehtinen, and Patrick Totzke. 2022. History-Deterministic Timed Automata. In *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland (LIPIcs, Vol. 243)*, Bartek Klin, Slawomir Lasota, and Anca Muscholl (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:21. DOI:https://doi.org/10.4230/LIPIcs.CONCUR.2022.14

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2007. *Introduction to automata theory, languages, and computation, 3rd Edition*. Addison-Wesley.

Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. 2016. Minimizing Regret in Discounted-Sum Games. In *Proceedings of CSL (LIPIcs, Vol. 62)*, Jean-Marc Talbot and Laurent Regnier (Eds.). 30:1–30:17. DOI:https://doi.org/10.4230/LIPIcs.CSL.2016.30

Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. 2017. Reactive synthesis without regret. *Acta Informatica* 54, 1 (2017), 3–39.

Simon Iosti and Denis Kuperberg. 2019. Eventually Safe Languages. In *Proceedings of DLT (Lecture Notes in Computer Science, Vol. 11647)*, Piotrek Hofman and MichałSkrzypczak (Eds.). Springer, 192–205. DOI:https://doi.org/10.1007/978-3-030-24886-4_14

Denis Kuperberg and Michał Skrzypczak. 2015. On Determinisation of Good-For-Games Automata. In *Proceedings of ICALP*. 299–310. DOI:https://doi.org/10.1007/978-3-662-47666-6_24

Orna Kupferman. 2023. Using the Past for Resolving the Future. (2023). To appear.

Orna Kupferman, Shmuel Safra, and Moshe Y Vardi. 2006. Relating word and tree automata. *Ann. Pure Appl. Logic* 138, 1-3 (2006), 126–146. DOI:https://doi.org/10.1016/j.apal.2005.06.009 Conference version in 1996.

Karoliina Lehtinen and Martin Zimmermann. 2022. Good-for-games ω-Pushdown Automata. *Log. Methods Comput. Sci.* 18, 1 (2022). DOI:https://doi.org/10.46298/lmcs-18(1:3)2022 Conference version at LICS 2020.

Christof Löding and Stefan Repke. 2013. Decidability Results on the Existence of Lookahead Delegators for NFA. In *Proc. of FSTTCS 2013*. 327–338. DOI:https://doi.org/10.4230/LIPIcs.FSTTCS.2013.327

Gila Morgenstern. 2003. Expressiveness results at the bottom of the ω-regular hierarchy. (2003). M.Sc. Thesis, The Hebrew University.

Aditya Prakash and K. S. Thejaswini. 2022. On History-Deterministic One-Counter Nets. DOI:https://doi.org/10.48550/ARXIV.2210.10084

Shmuel Safra and Moshe Y. Vardi. 1989. On ω-Automata and Temporal Logic. In *Proc. of STOC*. 127–137. DOI:https://doi.org/10.1145/73007.73019

Sven Schewe. 2020. Minimising Good-For-Games Automata Is NP-Complete. In *Proceedings of FSTTCS (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 182)*, Nitin Saxena and Sunil Simon (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 56:1–56:13. DOI:https://doi.org/10.4230/LIPIcs.FSTTCS.2020.56

Leslie G. Valiant. 1976. A Note on the Succinctness of Descriptions of Deterministic Languages. *Inf. Control.* 32, 2 (1976), 139–145. DOI:https://doi.org/10.1016/S0019-9958(76)90173-X