Explorable Automata

Emile Hazard
CNRS, LIP, ENS Lyon, France

Denis Kuperberg
CNRS, LIP, ENS Lyon, France

Abstract

We define the class of explorable automata on finite or infinite words. This is a generalization of History-Deterministic (HD) automata, where this time non-deterministic choices can be resolved by building finitely many simultaneous runs instead of just one. We show that recognizing HD parity automata of fixed index among explorable ones is in PTime, thereby giving a strong link between the two notions. We then show that recognizing explorable automata is ExpTime-complete, in the case of finite words or Büchi automata. Additionally, we define the notion of ω-explorable automata on infinite words, where countably many runs can be used to resolve the non-deterministic choices. We show that all reachability automata are ω-explorable, but this is not the case for safety ones. We finally show ExpTime-completeness for ω-explorability of automata on infinite words for the safety and co-Büchi acceptance conditions.

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

Keywords and phrases Nondeterminism, automata, complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.24

Related Version *Full Version*: `https://hal.archives-ouvertes.fr/hal-03669659/`

1 Introduction

In several fields of theoretical science, the tension between deterministic and non-deterministic models is a source of fundamental open questions, and has led to important lines of research. The most famous of this kind is the P vs NP question in complexity theory. This paper aims at further investigating the frontier between determinism and non-determinism in automata theory. Although Non-deterministic and Deterministic Finite Automata (NFA and DFA) are known to be equivalent in terms of expressive power, many subtle questions remain about the cost of determinism, and a deep understanding of non-determinism will be needed to solve them.

One of the approaches investigating non-determinism in automata is the study of History-Deterministic (HD) automata, introduced in [16] under the name Good-For-Game (GFG) automata. An automaton is HD if, when reading input letters one by one, its non-determinism can be resolved on-the-fly without any need to guess the future. This constitutes a model that is intermediary between non-determinism and determinism, and can sometimes bring the best of both worlds. Like deterministic automata, HD automata on infinite words retain good properties such as their soundness with respect to composition with games, making them appropriate for use in Church synthesis algorithms [16]. On the other hand, like non-deterministic automata, they can be exponentially more succinct than deterministic ones [19]. There is a very active line of research trying to understand the various properties of HD automata, see e.g. [1, 2, 7, 9, 20, 13, 24] for latest developments. The terminology *history-deterministic*, was introduced originally in the theory of regular cost functions [11]. The name "history-deterministic" corresponds to the above intuition of solving non-determinism on-the-fly, while the earlier name of "good-for-games" refers to sound composition with games. These two notions may actually differ in some quantitative frameworks, but coincide on boolean automata [8], and have been used interchangeably in most of the literature on the topic. In this paper, since we are mainly interested in resolving the non-determinism on-the-fly, we choose the HD denomination to emphasize this aspect[1].

The goal of this paper is to pursue this line of research by introducing and studying the class of explorable automata on finite and infinite words. The intuition behind explorability is to limit the amount of non-determinism required by the automaton to accept its language, in a more permissive way than HD automata. If $k \in \mathbb{N}$, an automaton is k-explorable if when reading input letters, it suffices to keep track of k runs to build an accepting one, if it exists. An automaton is explorable if it is k-explorable for some $k \in \mathbb{N}$. This can be seen as a variation on the notion of HD automaton, which corresponds to the case $k = 1$. The present work can be compared to [18], where a notion related to k-explorability (called *width*) is introduced and studied, see Section 2.3. In particular, some results of [18] also apply to k-explorability, notably ExpTime-completeness of deciding k-explorability of an NFA if k is part of the input. Surprisingly however, the techniques used in [18] are quite different from the ones we need here. This shows that fixing a bound k for the number of runs leads to very different problems compared to asking for the existence of such a bound.

One of the motivations to introduce the notion of explorability is to tackle one of the important open questions about HD automata: what is the complexity of deciding whether an automaton is HD? We explain in the following why explorability is relevant for this question, and show obstructions to some of our initial hopes in this direction.

Recognizing HD automata is known to be in PTime for Büchi [3] and co-Büchi [19] automata, but even for 3 parity ranks, the only known upper bound is ExpTime via the naive algorithm from [16]. We show how explorable automata can simplify this question: if the input automaton is explorable, then the problem becomes PTime for any fixed acceptance condition. Therefore, the question of recognizing HD automata can be shifted to: how hard is it to recognize explorable automata?

We then proceed to study the decidability and complexity of the explorability problem: deciding whether an input automaton on finite or infinite words is explorable. For this, we establish a connection with the population control problem studied in [4]. This problem asks, given an NFA with an arbitrary number of tokens in the initial state, whether a controller can choose input letters, thereby forcing every token to reach a designated state, even if tokens are controlled by an opponent. It is shown in [4] that the population control problem is ExpTime-complete, and we adapt their proof to our setting to show that the explorability problem is ExpTime-complete as well, already for NFAs. We also show that a direct reduction is possible, but at an exponential cost, yielding only a 2-ExpTime algorithm for the NFA explorability problem. In the case of infinite words, we adapt the proof to the Büchi case, thereby showing that the Büchi explorability problem is in ExpTime as well. We also remark that, as in [4], the number of tokens needed to witness explorability can go as high as doubly exponential in the size of the automaton.

This ExpTime-completeness result means that we unfortunately cannot directly use the intermediate notion of explorable automata to improve on the complexity of recognizing HD automata in full generality, as could have been the hope. However, there can also be some frameworks where we can guarantee to obtain an explorable automaton, and therefore easily decide whether it is HD. A recent example of this from [9] is detailed in Section 2.4. More generally, we believe that this explorability notion is of interest towards a better understanding of non-determinism in automata theory.

[1] This departs from earlier practices consisting in using HD and GFG in a way coherent with their contexts of introduction: HD for cost functions and GFG for boolean automata. Hence most of the papers cited here use GFG.

Notice that interestingly, from a model-checking perspective, our approach is dual to [4]: in the population control problem, an NFA is well-behaved when we can "control" it by forcing arbitrarily many runs to simultaneously reach a designated state, via an appropriate choice of input letters. On the contrary, in our approach, the input letters form an adversarial environment, and our NFA is well-behaved when its non-determinism is limited, in the sense that it is enough to spread finitely many runs to explore all possible behaviors.

On infinite words, we push further the notion of explorability, by remarking that for some automata, even following a countable number of runs is not enough. This leads to defining the class of *ω-explorable automata*, as those automata on infinite words where non-determinism can be resolved using countably many runs. We show that ω-explorable automata form a non-trivial class even for the safety acceptance condition (but not for reachability), and give an ExpTime algorithm recognizing ω-explorable automata, encompassing the safety and co-Büchi conditions. We also show ExpTime-hardness of this problem, by adapting the ExpTime-hardness proof of [4] to the setting of ω-explorability.

Summary of the contributions. We show that given an explorable parity automaton of fixed parity index, it is in PTime to solve its *HDness problem*, i.e. decide whether it is HD. The idea was already used in [3], and in [9] for quantitative automata. The algorithm used for Büchi HDness in [3] is conjectured to work for any acceptance condition (this is the "G_2 conjecture"), and it is in fact this algorithm that is shown here to work on any explorable parity automaton.

We show that given an NFA or Büchi automaton, it is decidable and ExpTime-complete to check whether it is explorable. Our proof of ExpTime-completeness for NFAs uses techniques developed in [4], where ExpTime-completeness is shown for the NFA population control problem. We generalize this result to ExpTime explorability checking for Büchi automata, requiring further adaptations. We also give a black box reduction using the result from [4]. This is enough to show decidability of the NFA explorability problem, but it yields a 2-ExpTime algorithm. As in [4], the ExpTime algorithm yields a doubly exponential tight upper bound on the number of tokens needed to witness explorability.

On infinite words, we show that any reachability automaton is ω-explorable, but that this is not the case for safety automata. We show that both the safety and co-Büchi ω-explorability problems are ExpTime-complete.

Related Works. Many works aim at quantifying the amount of non-determinism in automata. A survey by Colcombet [12] gives useful references on this question. Let us mention for instance the notion of ambiguity, which quantifies the number of simultaneous accepting runs. Similarly to [18], we can note that ambiguity is orthogonal to k-explorability. Remark however that our finite/countable/uncountable explorability hierarchy is reminiscent of the finite/polynomial/exponential ambiguity hierarchy (see e.g. [25]).

In [17], several ways of quantifying the non-determinism in automata are studied from the point of view of complexity, including notions such as the number of advice bits needed.

Another approach is studied in [23], where a measure of the maximum non-deterministic branching along a run is defined and compared to other existing measures.

Following the HD approach, a hierarchy of non-determinism and an analysis of this hierarchy via probabilistic models is given in [2].

The idea of k-explorability stems from the approach in [3], using games with tokens to tackle the HDness problem for Büchi automata. In this previous work, the idea of following a finite number of runs in parallel plays a central role in the proof. Remark however that the notion of explorability as studied here is stronger than what is needed in [3]. The k-explorability (and explorability) property was explicitly defined under the name k-History-Determinism in [9], as a proof tool to decide the HDness of LimInf and LimSup automata. The work [9] is part of a research effort to understand how partial determinism notions such as HDness play out in quantitative automata, see survey [5]. Our goal here is to investigate explorability as defining a natural class of automata on finite and infinite words, somehow giving it an "official status" not restricted to an intermediate proof tool.

History of this work. It is traditional in our community to present results as a finished product, abstracting away the path that led to it. This paragraph is an experiment: we believe that in addition to this practice, it can be interesting for the reader to have access to a history of how ideas developed.

The interest we took in the explorability notion originated in the fact that it makes deciding HDness much easier, and the hope was that by using this notion as an intermediate, we could obtain an algorithm improving on the ExpTime upper bound for deciding GFGness of parity automata of fixed index, e.g. to PSpace. As we described above, we ended up showing that this approach cannot yield an algorithm below ExpTime (at least not in full generality). However, although this was initially only a tool for this decision problem, explorability turns out to be a natural generalisation of HD automata, and an interesting class to study in itself. The first investigation of this notion, and in particular of its decidability, was the object of a short research internship by Milla Valnet under the supervision of the second author. It was expected that decidability of explorability would be a reachable goal for such a short internship, but it turned out that this was overly optimistic. The internship yielded preliminary results, and in particular was useful to introduce and study the notion of "coverability". This version of the problem does not take accepting conditions into account, but only asks that at any point of the run, every state that could be reached is actually occupied by a token. After the internship, we continued to use this coverability notion as a stepping stone towards an understanding of explorability. However, after more preliminary results and unsuccessful attempts at obtaining decidability, we discovered the connection between explorability and population control from [4], that rendered the intermediate notion of coverability useless for our purposes, and we then focused on exploiting that link. We chose to leave coverability out of the present exposition, as it feels like a "watered-down" version of explorability, but it could be useful in some contexts, hence we briefly mention it in this chronological account. Let us just informally state here that it is straightforward to modify our proofs in order to show that deciding whether an NFA is coverable is ExpTime-complete as well.

2 Explorable automata

2.1 Automata

If $i \leq j$ are integers, we will denote by $[i, j]$ the integer interval $\{i, i+1, \dots, j\}$. If S is a set, its cardinal will be denoted $|S|$, and its powerset $\mathcal{P}(S)$.

We work with a fixed finite alphabet Σ. We will use the following default notation for the components of an automaton $\mathcal{A}$: $Q_\mathcal{A}$ for its set of states, $q_0^\mathcal{A}$ for its initial state, $F_\mathcal{A}$ for its accepting states, $\Delta_\mathcal{A}$ for its set of transitions. Letter $\mathcal{A}$ in the components might be omitted when clear from context. We might also specify its alphabet by $\Sigma_\mathcal{A}$ instead of Σ for cases where different alphabets come into play. If $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $(p, a) \in Q \times \Sigma$, we will note $\Delta(p, a) = \{q \in Q, (p, a, q) \in \Delta\}$. If $X \subseteq Q$, we note $\Delta(X, a) = \bigcup_{p \in X} \Delta(p, a)$.

To simplify definitions, all automata in this paper will be assumed to be complete (by adding a rejecting sink state if needed). This means that for all $(p, a) \in Q \times \Sigma$, we assume $\Delta(p, a) \neq \emptyset$. The rejecting sink state will often be implicit in our constructions and examples.

We will consider non-deterministic automata on finite words (NFAs). A run of such an automaton on a word $a_1 a_2 \ldots a_n \in \Sigma^*$ is a sequence of states $q_0 q_1 \ldots q_n \in Q^*$ (q_0 being the initial state), such that, for all $i \in [0, n-1]$, we have $q_{i+1} \in \Delta(q_i, a_{i+1})$. Such a run is accepting if $q_n \in F$, i.e. if the run belongs to $Q^* F$. As usual, the language of an automaton $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that admit an accepting run.

We will also deal with automata on infinite words, and we recall here some of the standard acceptance conditions for such automata. A run on an infinite word $w \in \Sigma^\omega$ is now an infinite sequence of states, i.e. an element of Q^ω, starting in q_0 and following as before transitions of the automaton according to the letters of w. Such a run of Q^ω is accepting in a safety (resp. reachability, Büchi, co-Büchi) automaton if it belongs to F^ω (resp. $Q^* F Q^\omega$, $(Q^* F)^\omega$, $Q^* F^\omega$). States from F will be called Büchi states in Büchi automata, and states from $Q \setminus F$ will be called co-Büchi states in co-Büchi automata.

Finally, we also mention the parity acceptance condition: it uses a ranking function rk from Q to an interval of integers $[i, j]$. A run is accepting if the minimal rank appearing infinitely often is even (following the convention of [4]).

We will also use standard game notions such as arena, play and strategy. See e.g. [15] for a complete definition of these objects.

2.2 Games

A *game* $\mathcal{G} = (V_0, V_1, v_I, E, W)$ of infinite duration between two players 0 and 1 consists of: a finite set of *positions* V being a disjoint union of V_0 and V_1; an *initial position* $v_I \in V$; a set of *edges* $E \subseteq V \times V$; and a *winning condition* $W \subseteq V^\omega$. We will later use names more explicit than 0 and 1 for the players, describing their roles in the various games we will define.

A *play* is an infinite sequence of positions $v_0 v_1 v_2 \cdots \in V^\omega$ such that $v_0 = v_I$ and for all $n \in \mathbb{N}$, $(v_n, v_{n+1}) \in E$. A play $\pi \in V^\omega$ is *winning* for Player 0 if it belongs to W. Otherwise π is *winning* for Player 1.

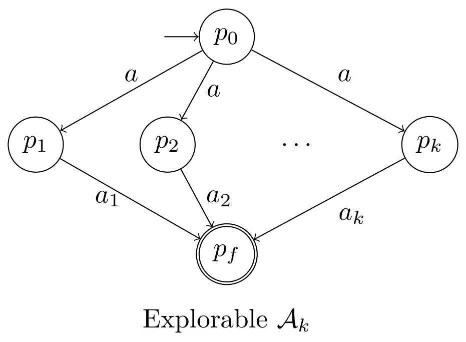
A *strategy* for Player 0 (resp. 1) is a function $\sigma_0 : V^* \times V_0 \to V$ (resp. $\sigma_1 : V^* \times V_1 \to V$), describing which edge should be played given the history of the play $u \in V^*$ and the current position $v \in V$. A strategy σ_P has to obey the edge relation, i.e. there has to be an edge in E from v to $\sigma_P(u, v)$. A play $\pi = v_0 v_1 v_2 \ldots$ is *consistent* with a strategy σ_P of a player P if for every n such that $v_n \in V_P$ we have $v_{n+1} = \sigma_P(v_0 \ldots v_{n-1}, v_n)$.

A strategy for Player 0 (resp. Player 1) is *positional* (or *memoryless*) if it does not use the history of the play, i.e. it can be seen as a function $V_0 \to V$ (resp. $V_1 \to V$).

We say that a strategy σ_P of a player P is *winning* if every play consistent with σ_P is winning for P. In this case, we say that P *wins* the game $\mathcal{G}$.

A game is *positionally determined* if exactly one of the players has a positional winning strategy in the game.

In the interest of readability, when describing games in the paper, we will not give explicit definitions of the sets V_0, V_1 and E, but give slightly more informal descriptions in terms of possible actions of players at each round. It is straightforward to build a formal description of the game from such a description.



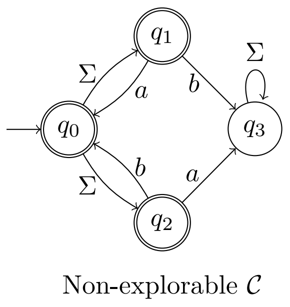


Figure 1 An explorable and a non-explorable automata.

2.3 Explorability

We start by introducing the *k-explorability game*, which is the central tool allowing us to define the class of explorable automata.

▶ **Definition 1** (k-explorability game)**.** *Consider a non-deterministic automaton $\mathcal{A}$ on finite or infinite words, and an integer k. The k-explorability game on $\mathcal{A}$ is played on the arena Q^k. The two players are called Determiniser and Spoiler, and they play as follows.*

- *The initial position is the k-tuple $S_0 = (q_0, \ldots, q_0)$.*
- *At step i from a position $S_{i-1} \in Q^k$, Spoiler chooses a letter $a_i \in \Sigma$, and Determiniser chooses $S_i \in Q^k$ such that for every token $l \in [0, k-1]$, $S_{i-1}(l) \xrightarrow{a_i} S_i(l)$ is a transition of $\mathcal{A}$ (where $S_i(l)$ stands for the l-th component in S_i).*

The play is won by Determiniser if for all $\beta \leq \omega$ such that the word $(a_i)_{1 \leq i < \beta}$ is in $\mathcal{L}(\mathcal{A})$, there is a token $l < k$ being accepted by $\mathcal{A}$, meaning that the sequence $(S_i(l))_{i<\beta}$ is an accepting run[2]. Otherwise, the winner is Spoiler.
We will say that $\mathcal{A}$ is k-explorable *if Determiniser wins the k-explorability game (i.e. has a winning strategy, ensuring the win independently of the choices of Spoiler).*
We will say that $\mathcal{A}$ is explorable *if it is k-explorable for some $k \in \mathbb{N}$.*

▶ **Example 2.** Consider the automata from Figure 1. The NFA $\mathcal{A}_k$ on alphabet $\{a, a_1, \ldots, a_k\}$ is k-explorable, but not $(k-1)$-explorable. It can easily be adapted to a binary alphabet, by replacing in the automaton $a_1, \ldots, a_k$ by distinct words of the same length.

On the other hand, the NFA $\mathcal{C}$ is a non-explorable NFA accepting all words on alphabet $\Sigma = \{a, b\}$. Indeed, Spoiler can win the k-explorability game for all k, by eliminating tokens one by one, choosing at each step the letter b if q_1 is occupied by at least one token, and the letter a otherwise.

▶ **Example 3.** The NFA $\mathcal{B}_k$ from Figure 2 with $3k+1$ states on alphabet $\Sigma = \{a, b\}$ is explorable, but requires 2^k tokens. Indeed, since when choosing the $2i^{\text{th}}$ letter Spoiler can always pick the state p_i or r_i containing the least amount of tokens to decide whether to play a or b, the best strategy for Determiniser is to split his tokens evenly at each q_i. This means he needs to start with 2^k tokens to end up with at least one token in q_k after a word of Σ^{2k}.

[2] This condition $\beta \leq \omega$ is actually accounting separately for the two cases of finite and infinite words, corresponding respectively to $\beta < \omega$ and $\beta = \omega$.

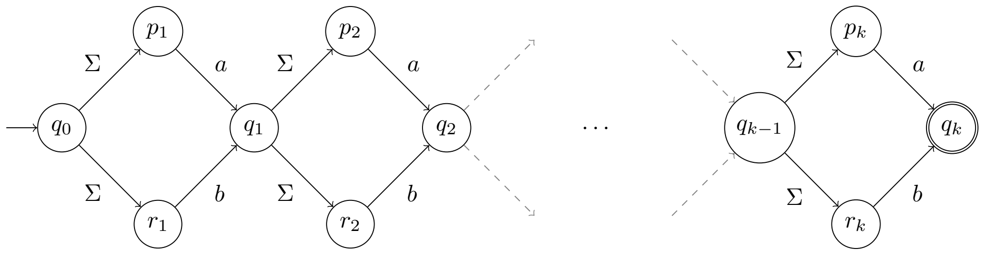


Figure 2 An explorable automaton $\mathcal{B}_k$ requiring exponentially many tokens.

Let us mention a few facts that follow from the definition of explorability:

▶ **Lemma 4.**
- *Any finite language is explorable.*
- *If $\mathcal{A}$ is k-explorable, then it is n-explorable for all $n \geq k$.*
- *If $\mathcal{A}$ is k-explorable and $\mathcal{B}$ is n-explorable, then*
  - *$\mathcal{A} \cup \mathcal{B}$ (with states $Q = \{q_0\} \cup Q_\mathcal{A} \cup Q_\mathcal{B}$) is $(k+n)$-explorable,*
  - *the union product $\mathcal{A} \times \mathcal{B}$ (with $F = (F_\mathcal{A} \times Q_\mathcal{B}) \cup (Q_\mathcal{A} \times F_\mathcal{B})$) is $\max(k,n)$-explorable,*
  - *the intersection product $\mathcal{A} \times \mathcal{B}$ (with $F = F_\mathcal{A} \times F_\mathcal{B}$) is (kn)-explorable.*

Proof. If $L(\mathcal{A})$ is finite, it is enough to take $k = |L(\mathcal{A})|$ tokens to witness explorability: for each $u \in L(\mathcal{A})$, the token t_u assumes that the input word is u and follows an accepting run of $\mathcal{A}$ over u as long as input letters are compatible with u. As soon as an input letter is not compatible with u, the token t_u is discarded and behaves arbitrarily for the rest of the play.

If $\mathcal{A}$ is k-explorable and $n \geq k$, then Determiniser can win the n-explorability game by using the same strategy with the first k tokens and making arbitrary choices with the $n-k$ remaining tokens.

If $\mathcal{A}$ and $\mathcal{B}$ are k- and n-explorable respectively, then Determiniser can use both strategies simultaneously with $k+n$ tokens in $\mathcal{A} \cup \mathcal{B}$, using k tokens in $\mathcal{A}$ and n tokens in $\mathcal{B}$. If the input word is in $\mathcal{A}$ (resp. $\mathcal{B}$), then the tokens playing in $\mathcal{A}$ (resp. $\mathcal{B}$) will win the play.

In the union product $\mathcal{A} \times \mathcal{B}$, it is enough to take $\max(k,n)$ tokens: if $0 \leq i < \min(k,n)$, the token number i follows the strategy of the token i in $\mathcal{A}$ on the first coordinate, and the strategy of the token i in $\mathcal{B}$ in the second one. If $\min(k,n) \leq i < \max(k,n)$, say wlog $k \leq i < n$, the token i follows an arbitrary strategy on the $\mathcal{A}$-component and the strategy of token i on the $\mathcal{B}$-component.

However, Determiniser may need up to kn tokens to play in $\mathcal{A} \times \mathcal{B}$ when the accepting set is $F_\mathcal{A} \times F_\mathcal{B}$: the token (i,j) will use the strategy of the token i in the k-explorability game of $\mathcal{A}$ together with the strategy of the token j in the n-explorability game of $\mathcal{B}$. This lower bound of kn cannot be improved: consider for instance the intersection product $\mathcal{A}_k \times \mathcal{A}_n$, where $\mathcal{A}_k, \mathcal{A}_n$ are from Example 2, using as alphabet the cartesian product of their respective alphabets: $\{a, a_1, a_2, \ldots, a_k\} \times \{a, b\}$. ◀

Notice that a similar notion was introduced in [18] under the name *width*. In [18], the emphasis is put on another version of the explorability game, where tokens can be duplicated, and $|Q|$ is an upper bound for the number of necessary tokens. In this work, we will on the contrary focus on non-duplicable tokens. However, some results of [18] still apply here. In particular the following holds:

▶ **Theorem 5** ([18, Rem. 6.9]). *Given an NFA $\mathcal{A}$ and an integer k, it is* ExpTime*-complete to decide whether $\mathcal{A}$ is k-explorable (even if we fix $k = |Q_\mathcal{A}|/2$).*

We aim here at answering a different question:

▶ **Definition 6** (Explorability problem). *The explorability problem is the question, given a non-deterministic automaton $\mathcal{A}$, of deciding whether it is explorable (i.e., whether there exists $k \in \mathbb{N}$ such that it is k-explorable).*

Questions: Is the explorability problem decidable? If yes, what is its complexity?
We will first give some motivation for this problem in Section 2.4.

2.4 Link with HD automata

An automaton $\mathcal{A}$ is History-Deterministic (HD) if it is 1-explorable, i.e. if there is a strategy $\sigma : \Sigma^* \to Q$ resolving the non-determinism based on the word read so far, with the guarantee that the run piloted by this strategy is accepting whenever the input word is in $L(\mathcal{A})$. See e.g. [6] for an introduction to HD automata.

We will give here an additional and stronger link between explorable and HD automata. In this part, we will mainly be interested in automata on infinite words.

The arguments in this section are already hinted at in [3], and made explicit in the context of quantitative automata in [9]. We sketch them here for completeness, in order to give some context for the relevance of the class of explorable automata.

One of the main open problems related to HD automata on infinite words is to decide, given a non-deterministic parity automaton, whether it is HD. For now, the problem is only known to be in PTime for co-Büchi [19] and Büchi [3] automata. Extending this result even to 3 parity ranks is still open, and only a naive ExpTime upper bound [16] is known in this case. The following result shows that explorability is relevant in this context:

▶ **Theorem 7.** *Given an explorable parity automaton $\mathcal{A}$ of fixed parity index, it is in PTime to decide whether it is HD.*

This is one of the motivations to get a better understanding of explorable automata. Indeed, if we can obtain an efficient algorithm for recognizing them, or if we are in a context guaranteeing that we are only dealing with explorable automata, this result shows that we can obtain an efficient algorithm for recognizing HD automata. Alternatively, even if membership to the class of explorable automata is provably hard to decide in general (as it will turn out), there can be some contexts where explorable automata are sufficient for the intended purposes. An example is given in [9], where it is shown that for LimSup and LimInf automata, Eve winning the game G_2 (defined below) implies that the automaton is explorable. Since Theorem 7 actually shows that Eve winning G_2 characterizes HDness for explorable automata, in this case it implies that the automaton is HD, as was shown in [9].

The rest of this section will be devoted to give a proof sketch of Theorem 7. See the long version (Appendix A.1) for formal details.

Let $\mathcal{A}$ be an explorable parity automaton, of fixed parity index $[i, j]$.

We briefly recall the definition of the k-token game $G_k(\mathcal{A})$ defined in [3], for an arbitrary $k \in \mathbb{N}$. At each round, Adam plays a letter $a \in \Sigma$, then Eve moves her token according to an a-transition, and finally Adam moves his k tokens according to a-transitions. Eve wins the play if her token builds an accepting run, or if all of Adam's tokens build a rejecting run.

We will prove that the game $G_2(\mathcal{A})$ is won by Eve if and only $\mathcal{A}$ is HD. Since $G_2(\mathcal{A})$ can be solved in PTime for fixed parity index [3], this is enough to conclude.

First, it is clear that if $\mathcal{A}$ is HD, then Eve wins $G_2(\mathcal{A})$ [3]: Eve can simply play her HD strategy with her token, ignoring Adam's tokens.

The interesting direction is the converse: we assume that Eve wins $G_2(\mathcal{A})$, and we show that under this assumption, $\mathcal{A}$ is necessarily HD. We use the following lemma:

▶ **Lemma 8** ([3, Thm. 14]). *Eve wins $G_2(\mathcal{A})$ if and only if Eve wins $G_k(\mathcal{A})$ for all $k \geq 2$.*

Since $\mathcal{A}$ is explorable, there is $k \in \mathbb{N}$ such that $\mathcal{A}$ is k-explorable. Let τ_k be a winning strategy for Determiniser in the k-explorability game of $\mathcal{A}$, and σ_k be a winning strategy for Eve in $G_k(\mathcal{A})$. We show that we can combine these two strategies to yield a HD strategy σ for $\mathcal{A}$. This proof follows the same idea as in [3] where the explorability hypothesis is not available, but $\mathcal{A}$ is assumed to be Büchi. The strategy σ will store k virtual tokens in its memory. When the automaton reads a new letter $a \in \Sigma$, these k tokens will be updated according to τ_k. Then the choice of σ will follow the strategy σ_k against these k tokens. Notice that the strategies τ_k and σ_k might use additional memory, but this is completely transparent in this proof scheme. If the input word is in $L(\mathcal{A})$, then by correctness of τ_k, one of the k virtual tokens will accept. Thus, by correctness of σ_k, the run chosen by σ will be accepting. Therefore, σ is a correct HD strategy, witnessing that $\mathcal{A}$ is HD. This concludes the proof sketch of Theorem 7.

3 Decidability and complexity of the explorability problem

In this section, we prove that the explorability problem is decidable and ExpTime-complete.

We start by showing in Section 3.1 decidability of the explorability problem for NFAs using the results of [4] as a black box. This yields an algorithm in 2-ExpTime. We give in Section 3.2 a polynomial reduction in the other direction, thereby obtaining ExpTime-hardness of the NFA explorability problem. To obtain a matching upper bound and show ExpTime-completeness, we use again [4], but this time we must "open the black box" and dig into the technicalities of their ExpTime algorithm while adapting them to our setting. We do so in Section 3.3, directly treating the more general case of Büchi automata.

3.1 2-ExpTime algorithm via a black box reduction

Let us start by recalling the population control problem (PCP) of [4].

▶ **Definition 9** (k-population game). *Given an NFA $\mathcal{B}$ with a distinguished target state $f \in Q_\mathcal{B}$, and an integer $k \in \mathbb{N}$, the k-population game is played similarly to the k-explorability game, only the winning condition differs: Spoiler wins if the game reaches a position where all tokens are in the state f.*

The PCP asks, given $\mathcal{B}$ and $f \in Q_\mathcal{B}$, whether Spoiler wins the k-population game for all $k \in \mathbb{N}$. Notice that this convention is opposite to explorability, where positive instances are defined via a win of Determiniser. The PCP is shown in [4] to be ExpTime-complete. We will present here a direct exponential reduction from the explorability problem to the PCP.

Let $\mathcal{A} = (\Sigma, Q_\mathcal{A}, q_0^\mathcal{A}, F_\mathcal{A}, \Delta_\mathcal{A})$ be an NFA. Our goal is to build an exponential NFA $\mathcal{B}$ with a distinguished state f such that $(\mathcal{B}, f)$ is a negative instance of the PCP if and only if $\mathcal{A}$ is explorable.

We choose $Q_\mathcal{B} = (Q_\mathcal{A} \times \mathcal{P}(Q_\mathcal{A})) \uplus \{f, \perp\}$, where $f, \perp$ are fresh sink states. The alphabet of $\mathcal{B}$ will be $\Sigma_\mathcal{B} = \Sigma \uplus \{a_{\text{test}}\}$, where a_{test} is a fresh letter.

The initial state of $\mathcal{B}$ is $q_0^\mathcal{B} = (q_0^\mathcal{A}, \{q_0^\mathcal{A}\})$. Notice that we do not need to specify accepting states in $\mathcal{B}$, as acceptance plays no role in the PCP.

We finally define the transitions of $\mathcal{B}$ in the following way:

- $(p, X) \xrightarrow{a} (q, \Delta_\mathcal{A}(X, a))$ if $a \in \Sigma$ and $q \in \Delta_\mathcal{A}(p, a)$,
- $(p, X) \xrightarrow{a_{\text{test}}} f$ if $p \notin F_\mathcal{A}$ and $X \cap F_\mathcal{A} \neq \emptyset$.
- $(p, X) \xrightarrow{a_{\text{test}}} \perp$ otherwise.

We aim at proving the following Lemma:

▶ **Lemma 10.** *For any $k \in \mathbb{N}$, $\mathcal{A}$ is k-explorable if and only if Determiniser wins the k-population game on $(\mathcal{B}, f)$.*

Notice that as long as letters of Σ are played, the second component of states of $\mathcal{B}$ evolves deterministically and keeps track of the set of reachable states in $\mathcal{A}$. Moreover, the letter a_{test} also acts deterministically on $Q_{\mathcal{B}}$. Therefore, the only non-determinism to be resolved in $\mathcal{B}$ is how the first component evolves, which amounts to building a run in $\mathcal{A}$. Thus, strategies driving tokens in $\mathcal{A}$ and $\mathcal{B}$ are isomorphic. It now suffices to observe that Spoiler wins the k-population game on $(\mathcal{B}, f)$ if and only if he has a strategy allowing to eventually play a_{test} while all tokens are in a state of the form (q, X) with $q \notin F_{\mathcal{A}}$ and $X \cap F_{\mathcal{A}} \neq \emptyset$. This is equivalent to Spoiler winning the k-explorability game of $\mathcal{A}$, since $X \cap F_{\mathcal{A}} \neq \emptyset$ witnesses that the word played so far is in $L(\mathcal{A})$.

This concludes the proof that $\mathcal{A}$ is explorable if and only if $(\mathcal{B}, f)$ is a negative instance of the PCP. So given an NFA $\mathcal{A}$ that we want to test for explorability, it suffices to build $(\mathcal{B}, f)$ as above, and use the ExpTime algorithm from [4] as a black box on $(\mathcal{B}, f)$. Since $\mathcal{B}$ is of exponential size compared to $\mathcal{A}$, we obtain the following result:

▶ **Theorem 11.** *The NFA explorability problem is decidable and in 2-ExpTime.*

3.2 ExpTime-hardness of NFA explorability

We will perform here an encoding in the converse direction: starting from an instance $(\mathcal{B}, f)$ of the PCP, we build polynomially an NFA $\mathcal{A}$ such that $\mathcal{A}$ is explorable if and only if $(\mathcal{B}, f)$ is a negative instance of the PCP.

It is stated in [4] that, without loss of generality, we can assume that f is a sink state in $\mathcal{B}$, and we will use this assumption here.

Let $\mathcal{C}$ be the 4-state automaton of Example 2, that is non-explorable and accepts all words on alphabet $\Sigma_{\mathcal{C}} = \{a, b\}$. Recall that, as an instance of the PCP, $\mathcal{B}$ does not come with an acceptance condition. We can assume that its accepting set is $F_{\mathcal{B}} = Q_{\mathcal{B}} \setminus \{f\}$.

We will take for $\mathcal{A}$ the product automaton $\mathcal{B} \times \mathcal{C}$ on alphabet $\Sigma_{\mathcal{A}} = \Sigma_{\mathcal{B}} \times \Sigma_{\mathcal{C}}$, with the union acceptance condition: $\mathcal{A}$ accepts whenever one of its components accepts. The transitions of $\mathcal{A}$ are defined as expected: $(p, p') \xrightarrow{a_1, a_2} (q, q')$ in $\mathcal{A}$ whenever $p \xrightarrow{a_1} q$ in $\mathcal{B}$ and $p' \xrightarrow{a_2} q'$ in $\mathcal{C}$.

Since $L(\mathcal{C}) = (\Sigma_{\mathcal{C}})^*$, we have $L(\mathcal{A}) = (\Sigma_{\mathcal{A}})^*$. The intuition for the role of $\mathcal{C}$ in this construction is the following: it allows us to modify $\mathcal{B}$ in order to accept all words, without interfering with its explorability status.

We claim that for any $k \in \mathbb{N}$, $\mathcal{A}$ is k-explorable if and only if Determiniser wins the k-population game on $(\mathcal{B}, f)$.

Assume that $\mathcal{A}$ is k-explorable, via a strategy σ. Then Determiniser can play in the k-population game on $(\mathcal{B}, f)$ using σ as a guide. In order to simulate σ, one must feed to it letters from $\Sigma_{\mathcal{C}}$ in addition to letters from $\Sigma_{\mathcal{B}}$ chosen by Spoiler. This is done by applying a winning strategy for Spoiler in the k-explorability game of $\mathcal{C}$. Assume for contradiction that, at some point, this strategy σ reaches a position where all tokens are in a state of the form (f, q) with $q \in Q_{\mathcal{C}}$. Since f is a sink state, when the play continues it will eventually reach a point where all tokens are in (f, q_3), where q_3 is the rejecting sink of $\mathcal{C}$. This is because we are playing letters from $\Sigma_{\mathcal{C}}$ according to a winning strategy for Spoiler in the k-explorability game of $\mathcal{C}$, and this strategy guarantees that all tokens eventually reach q_3 in $\mathcal{C}$. But this state (f, q_3) is rejecting in $\mathcal{A}$, and $L(\mathcal{A}) = (\Sigma_{\mathcal{A}})^*$, so this is a losing position for

Determiniser in the k-explorability game of $\mathcal{A}$. Since we assumed σ is a winning strategy in this game, we reach a contradiction. This means that following this strategy σ together with an appropriate choice for letters from $\Sigma_\mathcal{C}$, we guarantee that at least one token never reaches the sink state f on its $\mathcal{B}$-component. This corresponds to Determiniser winning in the k-population game on $(\mathcal{B}, f)$.

Conversely, assume that Determiniser wins in the k-population game on $(\mathcal{B}, f)$, via a strategy σ. The same strategy can be used in the k-explorability game of $\mathcal{A}$, by making arbitrary choices on the $\mathcal{C}$ component. As before, this corresponds to a winning strategy in the k-explorability game of $\mathcal{A}$, since there is always at least one token with $\mathcal{B}$-component in $F_\mathcal{B} = Q_\mathcal{B} \setminus \{f\}$. This completes the hardness reduction, and allows us to conclude:

▶ **Theorem 12.** *The NFA explorability problem is* ExpTime*-hard.*

▶ Remark 13. Using standard arguments, it is straightforward to extend Theorem 12 to ExpTime-hardness of explorability for automata on infinite words, using any of the acceptance conditions defined in Section 2.1.

Let us give some intuition on why we can obtain a polynomial reduction in one direction, but did not manage to do so in the other direction. Intuitively, the explorability problem is "more difficult" than the PCP for the following reason. In the PCP, Spoiler is allowed to play any letters, and the winning condition just depends on the current position. On the contrary, the winning condition of the k-explorability game mentions that the word chosen by Spoiler must belong to the language of the NFA. In order to verify this, we a priori need to append to the arena an exponential deterministic automaton for this language, and this is what is done in Section 3.1. This complicated winning condition is also the source of difficulty of recognizing HD parity automata.

3.3 ExpTime algorithm for Büchi explorability

▶ **Theorem 14.** *The explorability problem can be solved in* ExpTime *for Büchi automata (and all simpler conditions: NFA, safety, reachability).*

Due to space constraints, we will only sketch the proof of Theorem 14 here. A more detailed account is given in the long version (Appendix A.2).

The algorithm is adapted from the ExpTime algorithm for the PCP from [4]. We will recall here the main ideas of this algorithm, and describe how we adapt it to our setting.

Let $\mathcal{A}$ be an NFA, together with a target state f. The idea in [4] is to abstract the population game with arbitrary many tokens by a game called the *capacity game*. This game allows Determiniser to describe only the support of his set of tokens, i.e. the set of states occupied by tokens. The sequence of states obtained in a play can be analyzed via a notion of *bounded capacity*, in order to detect whether it actually corresponds to a play with finitely many tokens. This notion can be approximated by the more relaxed *finite capacity*, which is a regular property that is equivalent to bounded capacity in a context where games are finite-memory determined. This property of finite capacity can be verified by a deterministic parity automaton, yielding a parity game that can be won by Spoiler if and only if $(\mathcal{A}, f)$ is a positive instance of the PCP. Since this parity game has size exponential in $\mathcal{A}$, this yields an ExpTime algorithm for the PCP.

Here, we will perform the following tweaks to this construction. We now start with a Büchi automaton $\mathcal{A}$, and want to decide whether it is explorable.

First, we need to control that the infinite word played by Spoiler is in $L(\mathcal{A})$. This requires to build a deterministic parity automaton $\mathcal{D}$ recognising $L(\mathcal{A})$, and incorporate it into the arena. The size of $\mathcal{D}$ is exponential with respect to $\mathcal{A}$. We then follow [4] and build the capacity game augmented with $\mathcal{D}$. This time, a sequence of supports is winning if infinitely many of them contain an accepting state. We emphasize that we use here a particularity of the Büchi condition: observing the sequences of support sets of tokens is enough to decide whether one of the tokens follows an accepting run. The same particularity was used in [3], and was a crucial tool allowing to give a PTime algorithm for Büchi HDness. Since this modification still allows us to manipulate supports as simple sets, we can make use of the capacity game as before. See the long version, Appendix A.2, Remark 40 for an example showing that a naive adaptation of this construction to co-Büchi automata would not be correct.

Finally, we show that we can as in [4] obtain a parity game of exponential size characterizing explorability of $\mathcal{A}$, yielding the wanted ExpTime algorithm.

We also remark that, as in [4], this construction gives a doubly exponential upper bound on the number of tokens needed to witness explorability. Moreover, the proof from [4] that this is tight also stands here.

4 Explorability with countably many tokens

In this section, we look at the same problem of explorability of an automaton, but we now allow for infinitely many tokens. More precisely, we will redefine the explorability game to allow an arbitrary cardinal for the number of tokens, then consider decidability problems regarding that game. This notion will mainly be interesting for automata on infinite words.

4.1 Definition and basic results

The following definition extends the notion of k-explorability to non-integer cardinals:

▶ **Definition 15** (κ-explorability game)**.** *Consider an automaton $\mathcal{A}$ and a cardinal κ. The κ-explorability game on $\mathcal{A}$ is played on the arena $(Q_{\mathcal{A}})^\kappa$, between Determiniser and Spoiler. They play as follows.*

- *The initial position is S_0 associating q_0 to all κ tokens.*
- *At step i, from position S_{i-1}, Spoiler chooses a letter $a_i \in \Sigma$, and Determiniser chooses S_i such that for every token α, $S_{i-1}(\alpha) \xrightarrow{a_i} S_i(\alpha)$ is a transition in $\mathcal{A}$.*

The play is won by Determiniser if for all $\beta \leq \omega$ such that the word $(a_i)_{1\leq i<\beta}$ is in $\mathcal{L}(\mathcal{A})$, there is a token $\alpha \in \kappa$ building an accepting run, meaning that the sequence $(S_i(\alpha))_{i<\beta}$ is an accepting run. Otherwise, the winner is Spoiler.

We will say in particular that $\mathcal{A}$ is ω-explorable if Determiniser wins the game with ω tokens. We use here the notation ω for convenience, it should be understood as the countably infinite cardinal $\aleph_0$. We will however explicitly use the fact that such an amount of tokens can be labelled by $\mathbb{N}$, in order to describe strategies for Spoiler or Determiniser in the ω-explorability game. The following lemma gives a first few results on generalized explorability.

▶ **Lemma 16.**
- *Determiniser wins the explorability game on $\mathcal{A}$ with $|\mathcal{L}(\mathcal{A})|$ tokens.*
- *ω-explorability is not equivalent to explorability.*
- *There are non ω-explorable safety automata.*

Proof. For the first item, a strategy for Determiniser is to associate a token to each word of $\mathcal{L}(\mathcal{A})$ and to have it follow an accepting run for that word. Let us add a few details on the cardinality of $L(\mathcal{A})$. First, a dichotomy result has been shown in [22] (even in the

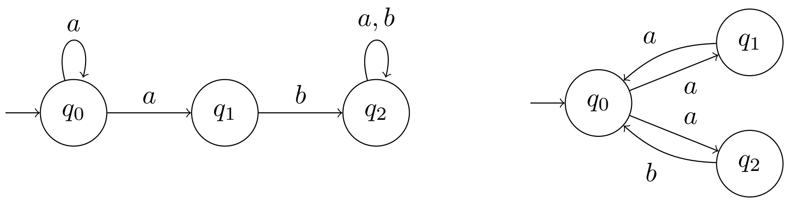


Figure 3 Two safety automata. Left: ω-explorable but not explorable. Right: not ω-explorable.

more general case of infinite trees): if $L(\mathcal{A})$ is not countable, then it has the cardinality of continuum, and this happens if and only if $L(\mathcal{A})$ contains a non ultimately periodic word. In this case, we can simply associate a token with every possible run. In the other case where $L(\mathcal{A})$ is countable, we have to associate an accepting run to each word, and this can be done without needing the Axiom of Countable Choice: a canonical run can be selected (e.g. lexicographically minimal).

We now want to prove that there are automata that are ω-explorable but not explorable. One such automaton is given in Figure 3 (left), where the rejecting sink state is omitted. Against any finite number of tokens, Spoiler has a strategy to eliminate them one by one, by playing a while Determiniser sends tokens to q_1, and b the first time q_1 is empty after the play of Determiniser. On the other hand, with tokens indexed by ω, Determiniser can keep the token 0 in q_0, and send token i to q_1 at step i. Those strategies are winning, which proves both non explorability and ω-explorability of the automaton.

The last item is proven by the second example from Figure 3. A winning strategy for Spoiler against countable tokens consists in labelling the tokens with integers, then targeting each token one by one (first token 0, then 1, 2, etc.). Each token is removed using the correct two-letters sequence (a, then b if the token is in q_1 or a if it is in q_2). With this strategy, every token is removed at some point, even if there might always be tokens in the game. ◀

The first item of Lemma 16 implies that the ω-explorability game only gets interesting when we look at automata over infinite words: since any language of finite words over a finite alphabet is countable, Determiniser wins the corresponding ω-explorability game. We will therefore focus on infinite words in the following.

Let us emphasize the following slightly counter-intuitive fact: in the ω-explorability game, it is always possible for Determiniser to guarantee that infinitely many tokens occupy each currently reachable state. However, even in a safety automaton, this is not enough to win the game, as it does not prevent that each individual token might be eventually "killed" at some point. As the following Lemma shows, this phenomenon does not occur in reachability automata on infinite words.

▶ **Lemma 17.** *Any reachability automaton is ω-explorable.*

Proof. Notice first that although the argument is very similar to the one for finite words, we cannot use exactly the same property: a reachability language can still be uncountable, so using one token per word of the language is not possible.

For every $w \in \Sigma^*$ such that there is a finite run ρ leading to an accepting state, Determiniser can use a single token following ρ. This token will accept all words of $w \cdot \Sigma^\omega$. Since Σ^* is countable, we only need countably many such tokens to cover the whole language, hence the result.

Let us give another equally simple view: a winning strategy for Determiniser in the ω-explorability game is to keep infinitely many tokens in each currently reachable state, as described above. Since acceptance in a reachability automaton is witnessed at a finite time, this strategy is winning. ◀

4.2 ExpTime algorithm for co-Büchi automata

We already know, from the example of Figure 3, that the result from Lemma 17 does not hold in the case of safety automata. However, we have the following decidability result, which talks about co-Büchi automata, and therefore still holds for safety automata as a subclass of co-Büchi.

▶ **Theorem 18.** *The ω-explorability of co-Büchi automata is decidable in* ExpTime.

To prove this result, we will use the following *elimination game.* $\mathcal{A}$ will from here on correspond to a co-Büchi (complete) automaton. We start by building a deterministic co-Büchi automaton $\mathcal{D}$ for $L(\mathcal{A})$ (e.g. using the breakpoint construction [21]).

▶ **Definition 19** (Elimination game)**.** *The elimination game is played on the arena $\mathcal{P}(Q_\mathcal{A}) \times Q_\mathcal{A} \times Q_\mathcal{D}$. The two players are named Protector and Eliminator, and the game proceeds as follows, starting in the position $(\{q_0^\mathcal{A}\}, q_0^\mathcal{A}, q_0^\mathcal{D})$.*

- *From position (B, q, p) Eliminator chooses a letter $a \in \Sigma$.*
- *If q is not a co-Büchi state, Protector picks a state $q' \in \Delta_\mathcal{A}(q, a)$.*
- *If q is a co-Büchi state, Protector picks any state $q' \in \Delta_\mathcal{A}(B, a)$. Such an event is called* elimination*.*
- *The play moves to position $(\Delta_\mathcal{A}(B, a), q', \delta_\mathcal{D}(p, a))$.*

Such a play can be written $(B_0, q_0, p_0) \xrightarrow{a_1} (B_1, q_1, p_1) \xrightarrow{a_2} (B_2, q_2, p_2) \ldots$, and Eliminator wins if infinitely many q_i and finitely many p_i are co-Büchi states.

Intuitively, what is happening in this game is that Protector is placing a token that he wants to protect in a reachable state, and Eliminator aims at bringing that token to a co-Büchi state while playing a word of $L(\mathcal{A})$. If Protector eventually manages to preserve his token from elimination on an infinite suffix of the play, he wins.

▶ **Lemma 20.** *The elimination game can be solved in polynomial time (in the size of the game).*

Proof. To prove this result, we simply need to note that the winning condition is a parity condition of fixed index. If we label the co-Büchi states p_i with rank 1, the co-Büchi states q_i with rank 2, and the others with 3, then take the lowest rank in (B_i, q_i, p_i) (ignoring B_i), Eliminator wins if and only if the inferior limit of ranks is even. As any parity game with 3 ranks can be solved in polynomial time [10], this is enough to get the result. ◀

We want to prove the equivalence between this game and the ω-explorability game to obtain Theorem 18.

▶ **Lemma 21.** *$\mathcal{A}$ is ω-explorable if and only if Protector wins the elimination game on $\mathcal{A}$.*

Proof. First, let us suppose that Eliminator wins the elimination game on $\mathcal{A}$. To build a strategy for Spoiler in the ω-explorability game of $\mathcal{A}$, we first take a function $f : \mathbb{N} \to \mathbb{N}$ such that for any $n \in \mathbb{N}$, $|f^{-1}(n)|$ is infinite (for instance f is described by the sequence $0,\ 0,1,\ 0,1,2,\ 0,1,2,3,\ldots$). The strategy for Spoiler will focus on sending token $f(0)$, then $f(1)$, then $f(2)$, etc. to a co-Büchi state.

Let σ be a memoryless winning strategy for Eliminator in the elimination game (recall that parity games do not require memory [14]). Spoiler will follow this strategy σ in the ω-explorability game, by keeping an imaginary play of the elimination game in his memory: $M = \mathcal{P}(Q_\mathcal{A}) \times Q_\mathcal{A} \times Q_\mathcal{D} \times \mathbb{N}$.

- At first, the memory holds the initial state $(\{q_0^{\mathcal{A}}\}, q_0^{\mathcal{A}}, q_0^{\mathcal{D}}, 0)$, and the current target is given by the last component: it is the token $f(0)$.
- From (B, q, p, n) Spoiler plays in both games the letter a given by $\sigma(B, q, p)$.
- Once Determiniser has played, Spoiler moves the memory to $(\Delta_{\mathcal{A}}(B, a), q', \delta_{\mathcal{D}}(p, a), n)$ where q' is the new position of the token $f(n)$, except if q was a co-Büchi state, in which case we move to $(\Delta_{\mathcal{A}}(B, a), q', \delta_{\mathcal{D}}(p, a), n+1)$ where q' is the new position of the token $f(n+1)$. We then go back to the previous step.

This strategy builds a play of the elimination game in the memory, that is consistent with σ. We know that σ is winning, which implies that the word played is in $\mathcal{L}(\mathcal{A})$, and that every $n \in \mathbb{N}$ is visited (each elimination increments n, and there are infinitely many of those). An elimination happening while the target is the token $f(n)$ corresponds, on the exploration game, to that token visiting a co-Büchi state. Ultimately this means that Determiniser did not provide any accepting run, while Spoiler did play a word from $\mathcal{L}(\mathcal{A})$, and therefore won.

Let us now consider the situation where Protector wins the elimination game, using some strategy τ. We want to build a winning strategy for Determiniser in the ω-explorability game. Similarly, this strategy will keep track of a play in the elimination game in its memory. Determiniser will maintain ω tokens in any reachable state, while focusing on a particular token which follows the path of the current target in the elimination game. When that token visits a co-Büchi state, we switch to the new token specified by τ.

Since τ is winning in the elimination game, either the word played by Spoiler is not in $\mathcal{L}(\mathcal{A})$, which ensures a win for Determiniser, or there are no eliminations after some point, meaning that the target token at that point never visits another co-Büchi state, which also implies that Determiniser wins. ◀

With Lemmas 20 and 21 we get a proof of Theorem 18, since the elimination game associated to $\mathcal{A}$ is of exponential size and can be built using exponential time.

4.3 ExpTime-hardness of the ω-explorability problem

▶ **Theorem 22.** *The ω-explorability problem for (any automaton model embedding) safety automata is* ExpTime*-hard.*

We give a quick summary of the proof in this section. The full proof can be found in the long version (Appendix A.3). The main idea will be to reduce the acceptance problem of a PSpace alternating Turing machine (ATM) to the ω-explorability problem of some automaton that we build from the machine. This reduction is an adaptation of the one from [4] showing ExpTime-hardness of the NFA population control problem (defined in Section 3.1).

The computation of an ATM can be seen as a game between two players, who respectively aim for acceptance and rejection of the input. These players influence the output by choosing the transitions when facing a non-deterministic choice, that can belong to either one of them.

Let us first describe the automaton built in [4]. In that reduction, the choices made by the ATM players are translated into choices for Determiniser and Spoiler. The automaton has two main blocks: one dedicated to keeping track of the machine's configuration, which we call Config, and another focusing on the simulation of the ATM choices, which we call Choices. In Config, there is no non-determinism: the tokens move following the transitions of the machine given as input to the automaton. In Choices, Determiniser can pick a transition by sending his token to the corresponding state, while Spoiler uses letters to pick his.

The automaton constructed this way will basically read a sequence of runs of the ATM. At each run, some tokens must be sent into both blocks. Reaching an accepting state of a run lets Spoiler send some tokens from Choices to his target state, specifically those whose choices for the transitions of the ATM were followed. He can then restart with the remaining tokens until all are in the target. This process will ensure a win for Spoiler if he has a winning strategy in the ATM game. If he does not, then Determiniser can use a strategy ensuring rejection in the ATM game to avoid the configurations where he loses tokens, provided he starts with enough tokens.

This equivalence between acceptance of the ATM and the automaton being a positive instance of the PCP provides the ExpTime-hardness of their problem.

In our setup, getting rid of tokens one by one is not enough: Spoiler needs to be able to target a specific token and send it to the target state (which is now the rejecting state $\perp$) in one run. If he can do that, repeating the process for every token, without omitting any, ensures his win. If he cannot, then Determiniser has a strategy to pick a specific token and preserving it from $\perp$, and therefore wins.

This is why we adapt our reduction to allow Spoiler to target a specific token, no matter where it chooses to go. To do so, we change the transitions so that winning a run lets Spoiler additionally send every token from Config into $\perp$. With that and the fact that he can already target a token in Choices, we get a winning strategy for Spoiler when the ATM is accepting.

If the ATM is rejecting, Spoiler is still able to send some tokens to $\perp$, but he no longer has that targeting ability, which is how Determiniser is able to build a strategy preserving a specific token to win. To ensure the sustainability of this method, Determiniser needs to keep ω additional tokens following his designated token, so that he always has ω tokens to spread into the gadgets every time a new run starts.

Overall, we are able to compute in polynomial time from the ATM a safety automaton that is ω-explorable if and only if the ATM rejects its input. Since acceptance of a polynomial space ATM is known to be ExpTime-hard, we obtain Theorem 22.

Conclusion

We introduced and studied the notions of explorability and ω-explorability, for automata on finite and infinite words. We showed that these problems are ExpTime-complete for Büchi condition in the first case and co-Büchi condition in the second case.

It is plausible that these results could be generalized to higher parity conditions, for instance by replacing the notion of support set by Safra trees, but this is outside the scope of this paper, and we leave this investigation for further research.

We showed that the original motivation of using explorability to improve the current knowledge on the complexity of the HDness problem for all parity automata cannot be directly achieved, since deciding explorability is at least as hard as HDness. Although this is a negative result, we believe it to be of importance. Moreover, some contexts naturally yield explorable automata, such as [9] where it leads to a PTime algorithm deciding the HDness of quantitative LimInf and LimSup automata. More generally, explorability is a natural property in the study of degrees of nondeterminism, and this notion could be used in other contexts as a middle ground between deterministic and non-deterministic automata.

References

1 Bader Abu Radi and Orna Kupferman. Minimization and canonization of GFG transition-based automata. *CoRR*, abs/2106.06745, 2021.

2 Bader Abu Radi, Orna Kupferman, and Ofer Leshkowitz. A hierarchy of nondeterminism. In *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

3 Marc Bagnol and Denis Kuperberg. Büchi good-for-games automata are efficiently recognizable. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

4 Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, Hugo Gimbert, and Adwait Amit Godbole. Controlling a population. *Log. Methods Comput. Sci.*, 15(3), 2019.

5 Udi Boker. Between deterministic and nondeterministic quantitative automata (invited talk). In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

6 Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Automata, Languages, and Programming – 40th International Colloquium, ICALP 2013*, Lecture Notes in Computer Science. Springer, 2013.

7 Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak. On the succinctness of alternating parity good-for-games automata. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

8 Udi Boker and Karoliina Lehtinen. History determinism vs. good for gameness in quantitative automata. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

9 Udi Boker and Karoliina Lehtinen. Token games and history-deterministic quantitative automata. In *Foundations of Software Science and Computation Structures – 25th International Conference, FOSSACS 2022*, Lecture Notes in Computer Science. Springer, 2022.

10 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 252–263. ACM, 2017.

11 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150, Berlin, 2009. Springer.

12 Thomas Colcombet. Forms of determinism for automata (invited talk). In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

13 Thomas Colcombet and Nathanaël Fijalkow. Universal graphs and good for games automata: New tools for infinite duration games. In *Foundations of Software Science and Computation Structures – 22nd International Conference, FOSSACS 2019*, Lecture Notes in Computer Science. Springer, 2019.

14 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991.

15 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

16 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *Computer Science Logic, 20th International Workshop, CSL 2006*, 2006.

17 Juraj Hromkovic, Juhani Karhumäki, Hartmut Klauck, Georg Schnitger, and Sebastian Seibert. Measures of nondeterminism in finite automata. *Electronic Colloquium on Computational Complexity (ECCC)*, 7, January 2000.

18 Denis Kuperberg and Anirban Majumdar. Computing the width of non-deterministic automata. *Log. Methods Comput. Sci. (LMCS)*, 15(4), 2019.

19 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015*, Lecture Notes in Computer Science. Springer, 2015.

20 Karoliina Lehtinen and Martin Zimmermann. Good-for-games ω-pushdown automata. In *LICS 2020: 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 689–702. ACM, 2020.

21 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoret. Comput. Sci.*, 32(3):321–330, 1984.

22 Damian Niwiński. On the cardinality of sets of infinite trees recognizable by finite automata. In *Mathematical Foundations of Computer Science 1991, 16th International Symposium, MFCS'91*, Lecture Notes in Computer Science. Springer, 1991.

23 Alexandros Palioudakis, Kai Salomaa, and Selim G. Akl. Worst case branching and other measures of nondeterminism. *Int. J. Found. Comput. Sci.*, 28(3):195–210, 2017.

24 Sven Schewe. Minimising good-for-games automata is NP-complete. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

25 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theor. Comput. Sci.*, 88(2):325–349, 1991.