



HAL
open science

Explorable Parity Automata

Emile Hazard, Olivier Idir, Denis Kuperberg

► **To cite this version:**

| Emile Hazard, Olivier Idir, Denis Kuperberg. Explorable Parity Automata. 2024. hal-04774282

HAL Id: hal-04774282

<https://hal.science/hal-04774282v1>

Preprint submitted on 8 Nov 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

EXPLORABLE PARITY AUTOMATA

EMILE HAZARD^a, OLIVIER IDIR^b, AND DENIS KUPERBERG ^a

e-mail address: emile.hazard@ens-lyon.fr

e-mail address: olivier.idir@irif.fr

e-mail address: denis.kuperberg@ens-lyon.fr

^a CNRS, LIP, ENS Lyon, France

^b IRIF, Université Paris-Cité, 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 parity automata up to index $[0, 2]$. 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 EXPTIME-completeness for ω -explorability of automata on infinite words for the safety and coBüchi acceptance conditions. We finally characterize the expressivity of (ω -)explorable automata with respect to the parity index hierarchy.

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 [HP06] under the name Good-For-Games (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 [HP06]. On the other hand, like non-deterministic automata, they can be exponentially more succinct

than deterministic ones [KS15]. There is a very active line of research trying to understand the various properties of HD automata, see e.g. [AK22, BKLS20, BL22, Cas23] for some of the recent developments. The terminology *history-deterministic*, was introduced originally in the theory of regular cost functions [Col09]. 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 [BL21], 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 aspect¹.

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 [KM19], where a notion related to k -explorability (called *width*) is introduced and studied, see Section 2.4. In particular, some results of [KM19] 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 [KM19] 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 [BK18] and coBüchi [KS15] automata, but even for 3 parity ranks, the only known upper bound is EXPTIME via the naive algorithm from [HP06]. 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 [BDG⁺19]. 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 [BDG⁺19] 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

¹This 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.

as well. We also remark that, as in [BDG⁺19], 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 [BL22] is detailed in Section 2.5.2. More generally, we believe that this explorability notion is of interest towards a better understanding of non-determinism in automata theory.

Notice that interestingly, from a model-checking perspective, our approach is dual to [BDG⁺19]: 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.

We also establish the expressivity of explorable automata, through reduction to or from deterministic parity automata. Surprisingly, the expressivity hierarchy of the explorable automata collapse, just as the hierarchy of non-determinist automata, albeit one step later. The general case is reached with parity-[1,3] explorable automata, which can recognize all regular languages.

On infinite words, we push further the notion of explorability, by remarking that for some automata, even following a countably infinite 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 coBüchi conditions. We also show EXPTIME-hardness of this problem, by adapting the EXPTIME-hardness proof of [BDG⁺19] 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 [BK18], and in [BL22] for quantitative automata. The algorithm used for Büchi HDness in [BK18] 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 a parity automaton with parities $\subseteq [0, 2]$, it is decidable and EXPTIME-complete to check whether it is explorable. We also study the expressivity in terms of recognized languages of the different parity classes of explorable automata. Our proof of EXPTIME-completeness for NFA explorability uses techniques developed in [BDG⁺19], where EXPTIME-completeness is shown for the NFA population control problem. We generalize this result to EXPTIME explorability checking for parity $[0, 2]$ automata, requiring further adaptations. We also give a black box reduction using the result from [BDG⁺19]. This is enough to show decidability of the NFA explorability problem, but it yields a 2-EXPTIME algorithm. As in [BDG⁺19], the EXPTIME algorithm yields a doubly exponential tight upper bound on the number of tokens needed to witness explorability.

We show that deciding the explorability of parity [1,3] automata amounts to deciding the explorability of automata in the general parity case.

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 coBüchi ω -explorability problems are EXPTIME-complete. We also show that the Büchi case corresponds to the general case : any non-deterministic parity automaton can be converted in PTIME to a Büchi automaton with same ω -explorability status.

Related Works. Many works aim at quantifying the amount of non-determinism in automata. A survey by Colcombet [Col12] 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 [KM19], 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.* [WS91]).

In [HKK⁺00], 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 [PSA17], 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 [AKL21].

The idea of k -explorability stems from the approach in [BK18], 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 [BK18]. The k -explorability (and explorability) property was explicitly defined under the name k -History-Determinism in [BL22], as a proof tool to decide the HDness of LimInf and LimSup automata. The work [BL22] is part of a research effort to understand how partial determinism notions such as HDness play out in quantitative automata, see survey [Bok22]. 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 third 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 acceptance 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 [BDG⁺19], 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. The first results were obtained during the PhD of Emile Hazard, and published in [HK23]. Some new results, namely EXPTIME algorithm for coBüchi and $[0, 2]$ -explorability, and expressivity results, were obtained during the internship of Olivier Idir. This paper, extending [HK23], aims at gathering what we currently know about explorable automata.

2. EXPLORABLE AUTOMATA

2.1. Preliminaries. 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 $\mathcal{P}(S)$.

2.2. Automata. 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. If the automaton is clear from context, the subscript/superscript \mathcal{A} might be omitted. 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 $\Delta(X, a) = \bigcup_{p \in X} \Delta(p, a)$. A transition (p, a, q) will often be noted $p \xrightarrow{a} q$.

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 $a_1 a_2 \dots a_n \in \Sigma^*$ is a sequence of transitions $\delta_1 \dots \delta_n \in \Delta^*$, such that there exists a sequence of states q_0, \dots, q_n with for all $i \in [1, n]$, $\delta_i = (q_{i-1}, a_i, q_i)$, (q_0 being the initial state). Such a run is accepting if $q_n \in F$. 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 $w = a_1 a_2 \dots \in \Sigma^\omega$ is now an infinite sequence of transitions $\delta_1, \delta_2, \dots$, *i.e.* an element of Δ^ω . As before there must exist an underlying sequence of state q_0, q_1, q_2, \dots with q_0 the initial state, such that for each $i \geq 1$, we have $\delta_i = (q_{i-1}, a_i, q_i)$.

The acceptance conditions safety, reachability, Büchi and coBüchi are defined with respect to an accepting subset of transitions $F \subseteq \Delta$. Here are the languages of accepting runs among Δ^ω , for these four acceptance conditions:

- Safety: F^ω
- Reachability: $\Delta^* F \Delta^\omega$
- Büchi: $(\Delta^* F)^\omega$
- coBüchi: $\Delta^* F^\omega$

Transitions from F will be called Büchi transitions in Büchi automata, and transition from $\Delta \setminus F$ will be called coBüchi transitions in coBüchi automata.

Finally, we will also use the parity acceptance condition: it uses a ranking function rk from Δ to an interval of integers $[i, j]$, called the *parity index* of the automaton. A run is accepting if the maximal rank appearing infinitely often is even.

For conciseness, we will simply write $[i, j]$ -*automaton* for a parity automaton using ranks from $[i, j]$. Remark that Büchi automata correspond to $[1, 2]$ -automata, and coBüchi automata to $[0, 1]$ -automata.

For all these acceptance conditions on infinite words, we will sometimes use state-based acceptance instead of transition-based when more convenient for our constructions. Recall that we can switch from transition-based to state-based with a doubling of the number of states, and the translation from state-based to transition-based can be done without changing the size of the automaton. These translations do not affect any of the explorability properties considered in this paper. See [Cas23] for details on the merits of transition-based acceptance conditions over the state-based ones.

2.3. 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 $v_0 v_1 v_2 \dots \in V^\omega$ 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 $\sigma_0 : V^* \times V_0 \rightarrow V$ (resp. $\sigma_1 : V^* \times V_1 \rightarrow V$), 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 $\pi = v_0 v_1 v_2 \dots$ is *consistent* with a strategy σ_P of a player P if for every n such that $v_n \in V_P$ we have $v_{n+1} = \sigma_P(v_0 \dots v_{n-1}, v_n)$.

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 $V_0 \rightarrow V$ (resp. $V_1 \rightarrow V$).

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 one of the players has a positional winning strategy in the game.

See *e.g.* [GTW02] for more details on games and strategies.

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 games from such a description.

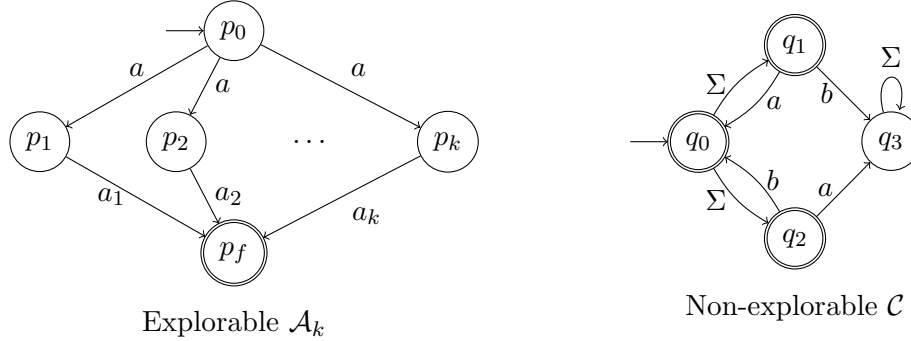


Figure 1: An explorable and a non-explorable automata

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

Definition 2.1 (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 $S_0 = (q_0, \dots, q_0)$.
- At step $i \geq 1$, 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 $l \in [1, k]$, $S_{i-1}(l) \xrightarrow{a_i} S_i(l)$ 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 $l \in [1, k]$ being accepted by \mathcal{A} , meaning that the sequence $(S_i(l))_{i < \beta}$ is an accepting run². 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 2.2. Consider the automata from Figure 1. The NFA \mathcal{A}_k on alphabet $\{a, a_1, \dots, a_k\}$ is k -explorable, but not $(k-1)$ -explorable. It can easily be adapted to a binary alphabet, by replacing in the automaton a_1, \dots, 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 2.3. 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 2^i -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^{2^k} .

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

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

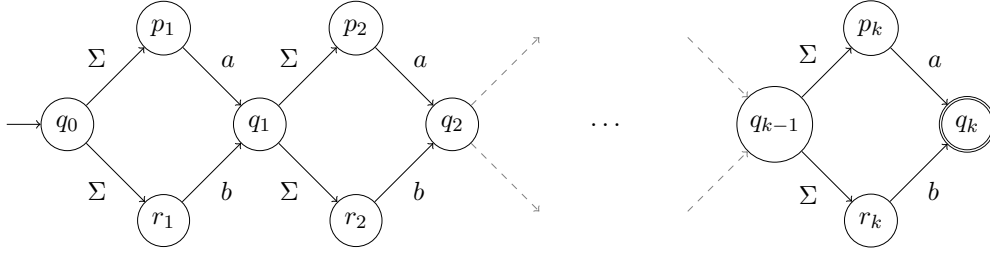


Figure 2: An explorable automaton \mathcal{B}_k requiring exponentially many tokens

Lemma 2.4.

- Any automaton for a 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 $1 \leq i \leq \min(k, n)$, 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 $\min(k, n) < i \leq \max(k, n)$, say wlog $k < i \leq n$, 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 2.2, using as alphabet the cartesian product of their respective alphabets: $\{a, a_1, a_2, \dots, a_k\} \times \{a, a_1, \dots, a_n\}$, or $\{a, b\}^2$ in their binary alphabet versions. \square

Notice that a similar notion was introduced in [KM19] under the name *width*. In [KM19], 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 [KM19] still apply here. In particular the following holds:

Theorem 2.5 ([KM19, 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 2.6 (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).

Another difference with the width setting from [KM19] is that here, some automata are explorable and some are not. Explorable automata can be seen as an intermediary model between deterministic and non-deterministic. Since deterministic and non-deterministic have very different expressive powers for each parity index, this naturally brings the question: what is the expressivity of explorable automata for each parity index ?

Questions: Is the explorability problem decidable? If yes, what is its complexity? How expressive are explorable automata for each parity index ?

2.5. Links with HD automata. An automaton \mathcal{A} is History-Deterministic (HD) if and only if it is 1-explorable, *i.e.* if there is a strategy $\sigma : \Sigma^* \rightarrow Q$ 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.* [BKKS13] for an introduction to HD automata.

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

2.5.1. Explorability in terms of HDness. Similarly to [KM19, Lem 3.5], we can express the k -explorability condition as a product automaton being HD.

Let \mathcal{A} be any non-deterministic parity automaton, and $k > 0$.

Definition 2.7. We denote by \mathcal{A}^k the union product of k copies of \mathcal{A} , *i.e.* the states are Q^k , and \mathcal{A}^k accepts if one of its copies follows an accepting run. The acceptance condition of \mathcal{A}^k is therefore the union of k parity conditions.

Lemma 2.8. *\mathcal{A} is k -explorable if and only if \mathcal{A}^k is HD.*

Proof. Winning strategies for Determinizer in the k -explorability game of \mathcal{A} are in bijection with winning strategies of Determinizer in the 1-explorability game of \mathcal{A}^k . \square

2.5.2. Recognizing HD automata among explorable ones. In this section we give some motivation for considering explorable automata.

The arguments in this section are already hinted at in [BK18], and made explicit in the context of quantitative automata in [BL22]. We give them here for completeness, in order to provide 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 coBüchi [KS15] and Büchi [BK18] automata. Extending this result even to 3 parity ranks is still open, and only a naive EXPTIME upper bound [HP06] is known in this case. The following result shows that explorability is relevant in this context:

Theorem 2.9. *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 [BL22], 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 2.9 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 [BL22].

Let \mathcal{A} be an explorable $[i, j]$ -automaton.

We briefly recall the definition of the k -token game $G_k(\mathcal{A})$ defined in [BK18], 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.

First, it is clear that if \mathcal{A} is HD, then Eve wins $G_2(\mathcal{A})$ [BK18]: 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 2.10 ([BK18, Thm. 14]). *Eve wins $G_2(\mathcal{A})$ if and only if Eve wins $G_k(\mathcal{A})$ for all $k \geq 2$.*

Here we will combine this general result with the following lemma, that is specific to explorable automata:

Lemma 2.11. *If \mathcal{A} is k -explorable and Eve wins $G_k(\mathcal{A})$, then \mathcal{A} is HD.*

Proof. Let us note $Q = Q_{\mathcal{A}}$ the set of states of \mathcal{A} . We will build an explicit strategy witnessing that \mathcal{A} is HD.

Let τ_k be a winning strategy for Determiniser in the k -explorability game of \mathcal{A} , and σ_k a winning strategy for Eve in $G_k(\mathcal{A})$.

Let us explicit in detail the shape of these strategies. The strategy τ_k has access to the history of the play in the k -explorability game, and must decide on a move for Determiniser. Notice that in absence of memory limitations, it is always enough to know the history of the opponent's moves (here the letters of Σ played so far), since this allows to compute the answer of Determiniser at each step, and therefore build a unique play. Thus, we can take for τ_k a function $\Sigma^* \rightarrow Q^k$. If the word played so far is $u \in \Sigma^*$, the tuple of states reached by the k tokens moved according to τ_k is $\tau_k(u) \in Q^k$. In particular $\tau_k(\varepsilon) = (q_0^A, \dots, q_0^A)$.

If $w = a_1 a_2 \dots \in \Sigma^\omega$, and $i \in \mathbb{N}$, let us note $(q_{w,1}^i, \dots, q_{w,k}^i) = \tau_k(a_1 \dots a_i)$. That is $q_{w,j}^i$ is the state reached by the j^{th} token after i steps in the run induced by τ_k and w . If $j \in [1, k]$, let us note $\rho_{w,j}$ the infinite run $q_{w,j}^0 q_{w,j}^1 q_{w,j}^2 \dots$, followed by the j^{th} token in this play. Since τ_k is a winning strategy in the k -explorability game of \mathcal{A} , we have the guarantee that for all $w \in L(\mathcal{A})$, there exists $j \in [1, k]$ such that $\rho_{w,j}$ is accepting.

If $u = a_1 \dots a_n \in \Sigma^*$ is a finite word, we define $\tau_k'(u) = (\tau_k(\varepsilon), \tau_k(a_1), \tau_k(a_1 a_2) \dots, \tau_k(u))$, this is a description of the k partial runs induced by τ_k on u .

Let us now turn to the strategy σ_k of Eve in $G_k(\mathcal{A})$. The type of this strategy is $\sigma_k : \Sigma^* \times (Q^k)^* \rightarrow Q$. Indeed, this time, the history of Adam's moves must contain his choice of letters together with his choices of positions for his k tokens. So $\sigma_k(u, \gamma)$ gives the state reached by Eve's unique token after a history (u, γ) for the moves of Adam. Notice that at each step, Eve must move before Adam in this game $G_k(\mathcal{A})$, so γ does not contain the choice of Adam on the last letter of u . This means that except for $u = \varepsilon$, we can always assume $|u| = |\gamma| + 1$ in a history (u, γ) .

We have the guarantee that if Adam plays an infinite word w together with runs ρ_1, \dots, ρ_k on w , such that at least one of these runs is accepting, then the run yielded by σ_k against $(w, (\rho_1, \dots, \rho_k))$ is accepting.

We finally define the HD strategy σ for \mathcal{A} , of type $\Sigma^* \rightarrow Q$, by induction: $\sigma(\varepsilon) = q_0^A$, and $\sigma(ua) = \sigma_k(ua, \tau_k'(u))$.

This amounts to playing the strategy σ_k in $G_k(\mathcal{A})$, against Adam playing a word w and moving his k tokens according to the strategy τ_k against w . If the infinite word $w = a_1 a_2 \dots$ chosen by Adam is in $L(\mathcal{A})$, then by correctness of τ_k one of the k runs $\rho_{w,1}, \dots, \rho_{w,k}$ yielded by τ_k is accepting. Hence, by correctness of σ_k , the run $\sigma(\varepsilon)\sigma(a_1)\sigma(a_1 a_2)$ yielded by σ (based on σ_k) is accepting. This concludes the proof that σ is a correct HD strategy for \mathcal{A} , witnessing that \mathcal{A} is HD. \square

Combining the results of this section, we obtain that if \mathcal{A} is explorable, then \mathcal{A} is HD if and only if Eve wins $G_2(\mathcal{A})$. It remains to show that the winner of $G_2(\mathcal{A})$ can be computed in PTIME, for fixed parity index of \mathcal{A} . This is already stated without proof in the conclusion of [BK18], but let us explicit it here for completeness, thereby achieving the proof of Theorem 2.9.

Lemma 2.12. *For parity automata of fixed parity index $[i, j]$, it is in PTIME to decide the winner of $G_2(\mathcal{A})$.*

Proof. The arena of $G_2(\mathcal{A})$ can be formalized as $Q^3 \cup (Q^3 \times \Sigma \times \{E, A\})$. In a state $(p, q_1, q_2) \in Q^3$, it is Adam's turn to choose a letter, so he can move to any (p, q_1, q_2, a, E) with $a \in \Sigma$. The E means that it is Eve's turn to choose a transition. In such a state (p, q_1, q_2, a, E) , Eve must choose a transition $p \xrightarrow{a} p'$, moving to a global position (p', q_1, q_2, a, A) . It is now Adam's turn to move his two tokens according to transitions $q_1 \xrightarrow{a} q_1'$ and $q_2 \xrightarrow{a} q_2'$, going to a global position (p', q_1', q_2') in the game. So the arena of the game is of size $O(n^3|\Sigma|)$, with $n = |Q|$ the size of \mathcal{A} . It is known that parity games of fixed index can be solved in PTIME [McN93, Thm 5.4 and 5.5]. However, the game $G_2(\mathcal{A})$ is not a parity game, since the winning condition is of the form $W = W_1 \vee (W_2 \wedge W_3)$, where W_1 is a parity condition expressing that Eve's token follows an accepting run, and W_2, W_3 are parity conditions on Adam's tokens, asking that they follow rejecting runs. Let $\Gamma = [i, j]^3$ be the alphabet used by these simultaneous parity conditions, each one using one of the components of the alphabet. We can view a play in $G_2(\mathcal{A})$ as outputting a word $\alpha \in \Gamma^\omega$, such that Eve wins the play if and only if $\alpha \in W$. Let \mathcal{D} be a deterministic parity automaton recognizing the language W on alphabet Γ . Notice that since $[i, j]$ is fixed, \mathcal{D} is a fixed parity automaton, and its size is a constant, not depending on n or Σ . We can now compose the game $G_2(\mathcal{A})$ with \mathcal{D} , to obtain a game G' , on arena $G_2(\mathcal{A}) \times \mathcal{D} \times ([i, j] \cup \{-\})$ (we abuse notation here and use $G_2(\mathcal{A})$ and \mathcal{D} to denote their sets of positions/states). This composition will work as follows:

- From a position $((p, q_1, q_2, a, E), q_{\mathcal{D}}, -)$, when Eve chooses a successor position in $G_2(\mathcal{A})$ according to a transition $\delta = (p, a, p')$, the game G' will move to $((p', q_1, q_2, a, A), q_{\mathcal{D}}, \text{rk}(\delta))$, i.e. remembering in the last component the parity rank seen by Eve's token.
- From a position $((p', q_1, q_2, a, A), q_{\mathcal{D}}, k)$, when Adam chooses a successor position in $G_2(\mathcal{A})$ according to transitions $\delta_1 = (q_1, a, q'_1)$ and $\delta_2 = (q_2, a, q'_2)$, the game G' will move to $((p', q'_1, q'_2), \delta_{\mathcal{D}}(q_{\mathcal{D}}, (k, \text{rk}(\delta_1), \text{rk}(\delta_2))), -)$, i.e. the automaton \mathcal{D} will deterministically advance according to the parity ranks seen on the 3 tokens.
- Eve's winning condition in G' is the parity condition of \mathcal{D} .

Such a composition of a game with a deterministic automaton is standard (up to the bookkeeping needed here to remember Eve's transition rank), and we obtain that G' and $G_2(\mathcal{A})$ have same winner: the winning condition of $G_2(\mathcal{A})$ is simply taken care of by \mathcal{D} in G' , allowing to simplify the winning condition of the game. Since \mathcal{D} is of fixed size, G' is still of polynomial size in \mathcal{A} , and it is a parity game, so we can decide its winner in PTIME. This completes the description of the PTIME algorithm to decide the winner of $G_2(\mathcal{A})$: build the game G' by composing $G_2(\mathcal{A})$ with a fixed deterministic parity automaton \mathcal{D} , and solve the game G' . \square

3. EXPRESSIVITY OF EXPLORABLE AUTOMATA

In this section, we ask the following question: what does the parity expressivity hierarchy look like for explorable automata? Recall that for deterministic automata, this hierarchy is strict, i.e. adding parity ranks allows recognizing more language. On the contrary, for non-deterministic automata, the hierarchy collapses at the Büchi level: any ω -regular language can be recognized by a non-deterministic Büchi automaton.

Finally, let us recall a classical result on expressivity of HD automata. We will also very briefly sketch its proof, as this will be useful in the following.

Lemma 3.1. [BKKS13] *For any parity index $[i, j]$, HD $[i, j]$ -automata recognize the same languages as deterministic $[i, j]$ -automata.*

Proof. (Sketch) The HD strategy can always be chosen as using a finite memory M . This memory M can be incorporated into the states of the automaton, making it deterministic, without changing its acceptance condition. \square

We will show in this section that explorable automata have an expressivity that is initially akin to the one of deterministic automata, but surprisingly the parity hierarchy collapses at the level $[1, 3]$, i.e. any ω -regular language can be recognized by an explorable $[1, 3]$ -automaton.

Let us start with the Büchi case to show some of the behaviours involved.

Lemma 3.2. *Languages recognized by explorable Büchi automata are equal to the languages recognized by deterministic Büchi automata.*

Proof. Notice that the converse inclusion is straightforward: if \mathcal{L} is recognized by a deterministic Büchi automaton \mathcal{D} , then \mathcal{D} is 1-explorable.

For the direct sense: Let L be a language recognized by a k -explorable Büchi automaton \mathcal{A} . We will build a deterministic Büchi automaton recognizing L . Let \mathcal{A}^k be the union product automaton from Definition 2.7. By Lemma 2.8, \mathcal{A}^k is HD. Moreover, \mathcal{A}^k can easily be turned into a Büchi automaton: we can exploit the fact the union of finitely many

Büchi conditions is a Büchi condition, by considering that any transition that is Büchi on some component is Büchi globally. This does not change the accepting status of any run, so the resulting Büchi automaton is still HD, using the same witness strategy as \mathcal{A}^k . Thus L is recognized by a HD Büchi automaton.

As recalled earlier, for any parity index $[i, j]$, HD $[i, j]$ -parity automata have same expressivity as deterministic $[i, j]$ -automata [BKKS13]. This is in particular true at the Büchi level, so there exists a deterministic Büchi automaton \mathcal{D} recognizing L . Notice that in the particular case of Büchi condition, this deterministic Büchi automaton can be guaranteed to be polynomial-size with respect to the HD Büchi automaton [KS15], and can also be obtained in PTIME [AJP24]. \square

The above proof can easily be adapted to get the following lemma:

Lemma 3.3. *Languages recognized by explorable safety (respectively reachability) automata are equal to the languages recognized by deterministic safety (respectively reachability) automata.*

We will now generalize this to $[0, 2]$ -automata:

Lemma 3.4. *Languages recognized by explorable $[0, 2]$ -automata are equal to the languages recognized by deterministic $[0, 2]$ -automata.*

Proof. Just as in the above proof, the converse inclusion is straightforward, as a deterministic automaton is always explorable. Let L be a language recognized by some k -explorable $[0, 2]$ -automaton \mathcal{A} . Similarly as above, this means that the union product \mathcal{A}^k is HD, and this is witnessed by a HD strategy with finite memory M . This allows us to build a deterministic automaton \mathcal{B} with states $Q^k \times M$, where the acceptance condition only depends on the Q^k component, and is a union of k $[0, 2]$ -conditions. In order to obtain a deterministic $[0, 2]$ -automaton, we need to compose \mathcal{B} with a deterministic $[0, 2]$ -automaton \mathcal{C} on alphabet $\Gamma := [0, 2]^k$, that accepts an infinite word if and only if one of its k -components is $[0, 2]$ -accepting. This can be obtained, by combining the well-known *breakpoint construction* on the $[0, 1]$ part, with the construction from Lemma 3.2, with ranks 2 playing the role of Büchi states.

Intuitively, the automaton will remember which components has seen a 1 since the last reset. Seeing a 2 anywhere causes to produce a 2 globally, and reset the memory. On intervals without any 2, if a 1 has been seen on every component, the automaton produces a 1 globally and resets its memory. The automaton $\mathcal{C} = (\Gamma, Q_{\mathcal{C}}, q_0^{\mathcal{C}}, \delta_{\mathcal{C}})$ can be built as follows:

- $Q_{\mathcal{C}} = \{0, 1\}^k$
- $q_0^{\mathcal{C}} = (0, 0, \dots, 0)$
- We directly label the transitions of \mathcal{C} by their parity rank, by giving a transition function $\delta_{\mathcal{C}} : (Q_{\mathcal{C}} \times \Gamma) \rightarrow (Q_{\mathcal{C}} \times [0, 2])$. It is defined as

$$\delta_{\mathcal{C}}((a_1, \dots, a_k), (b_1, \dots, b_k)) = \begin{cases} (q_0^{\mathcal{C}}, 2) & \text{if some } b_i \text{ is } 2, \\ (q_0^{\mathcal{C}}, 1) & \text{otherwise if for all } i, a_i = 1 \text{ or } b_i = 1, \\ ((a_i \vee b_i)_{1 \leq i \leq k}, 0) & \text{otherwise.} \end{cases}$$

Indeed, it is straightforward to verify that a run of \mathcal{C} is $[0, 2]$ -accepting if and only if this is the case for one of the k components of its input word. This means that \mathcal{C} deterministically translate the union of k $[0, 2]$ -conditions into one $[0, 2]$ -condition.

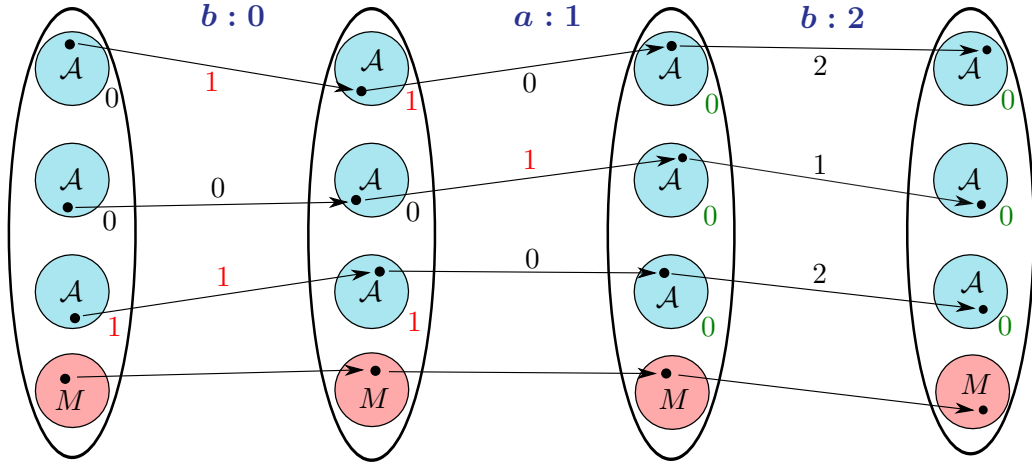


Figure 3: The composition of \mathcal{C} with $\mathcal{B} = \mathcal{A}^k \times M$ (here with $k = 3$). Internal ranks/memory states labelled 1 are represented in red, until the reset to $q_0^{\mathcal{C}}$ (in green) is performed when a 1 is reached on each component. Input letters and global ranks of the resulting automaton are shown on top.

Similarly to what was done in Lemma 2.12, it now suffices to compose \mathcal{B} with \mathcal{C} , i.e. having \mathcal{C} read the ranks output by \mathcal{B} , and using the $[0, 2]$ acceptance condition of \mathcal{C} . This yields a deterministic $[0, 2]$ -automaton recognizing L , represented in Figure 3. \square

We finally get to the general case with $[1, 3]$ -parity:

Theorem 3.5. *Let L be any ω -regular language, there exists an explorable $[1, 3]$ -automaton that recognizes L .*

Proof. Without loss of generality, the language L can be recognized by some deterministic $[1, d]$ -automaton \mathcal{A} with d even. We will build an explorable $[1, 3]$ -automaton recognizing L . For any given accepting run of \mathcal{A} , there is a unique even $l \in [1, d]$ such that l is the biggest priority encountered infinitely often. Conversely, if the run is rejecting, there exists no such even l . We will use this property to build an explorable automaton based on \mathcal{A} .

For l even, we define \mathcal{A}_l as the copy of \mathcal{A} where all priorities $< l$ are replaced with 1, all priorities $> l$ are replaced with 3, and all priorities equal to l are replaced with 2. \mathcal{A}_l is thus a deterministic $[1, 3]$ -automaton. It has the notable property that a word w is accepted by \mathcal{A}_l if and only if it is accepted by \mathcal{A} with highest priority l . Therefore \mathcal{A} is equivalent to the union of the $\{\mathcal{A}_l \mid l \text{ even} \in [1, d]\}$. We thus build the automaton \mathcal{A}' where the initial state branches non-deterministically via ε -transitions towards all the different \mathcal{A}_l for all even $l \in [1, d]$. This automaton is non-deterministic, of parity index $[1, 3]$, and recognizes exactly the words recognized by \mathcal{A} . It is $\frac{d}{2}$ -explorable, as this is the maximum number of tokens needed to place one in each \mathcal{A}_l at the start, after which their progression becomes deterministic. \square

This construction can actually be applied to non-deterministic automata as well, yielding the following result:

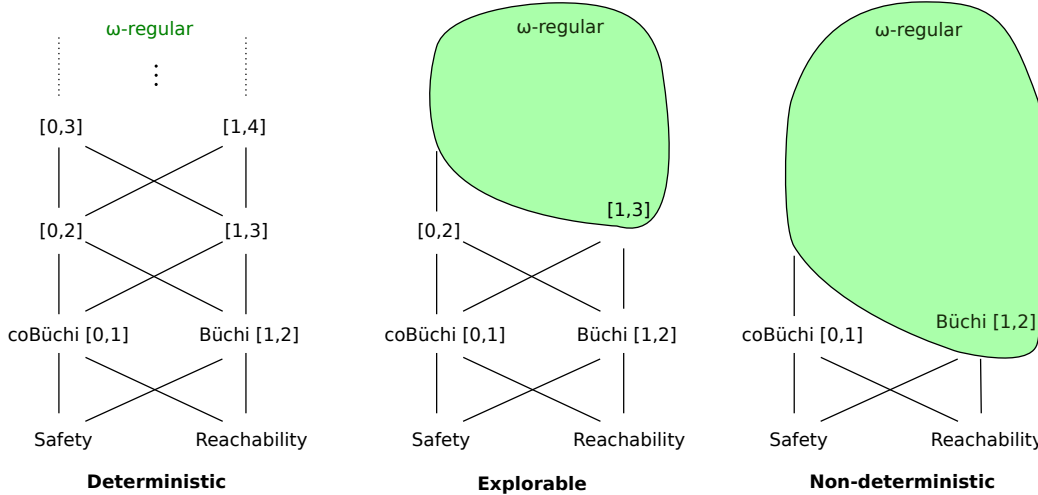


Figure 4: The parity hierarchy of languages recognized by the Deterministic/Explorable/Non-deterministic automata. Classes not included in the green collapse region always match their deterministic counterpart.

Lemma 3.6. *Let \mathcal{A} be a non-deterministic $[1, d]$ -automaton with d even. We can build in PTIME a $[1, 3]$ -automaton \mathcal{A}' recognizing $\mathcal{L}(\mathcal{A})$, such that \mathcal{A}' is explorable if and only if \mathcal{A} is explorable.*

Proof. We use the same construction as in the proof of Theorem 3.5 above. This construction is clearly PTIME in the size of \mathcal{A} , and produces an automaton \mathcal{A}' of size $|\mathcal{A}| \cdot \frac{d}{2}$. Moreover, language equivalence still holds: a word is accepted in \mathcal{A} if and only if it is accepted in at least one of the \mathcal{A}_l if and only if it is accepted in \mathcal{A}' . We now need to show the equivalence between explorability of \mathcal{A} and explorability of \mathcal{A}' .

If \mathcal{A} is k -explorable, then it suffices to initially send k tokens to each copy \mathcal{A}_l and from there use the k -explorability strategy in each copy locally. We get that if the input word can be accepted with some token i via the k -explorability strategy in \mathcal{A} with highest parity l , then the corresponding token is accepting in \mathcal{A}_l , and \mathcal{A}' is thus $\frac{kd}{2}$ -explorable.

If \mathcal{A}' is k -explorable, the behaviour of each of the k tokens can be projected to \mathcal{A} , giving a candidate strategy for k -explorability of \mathcal{A} . This is indeed a valid strategy, as if a token is accepting in \mathcal{A}_l , it is also accepting in \mathcal{A} . We can conclude that \mathcal{A} is k -explorable as well. \square

We thus obtain the hierarchy of languages recognized by explorable automata, represented in Figure 4. This picture will be completed in Section 5.4, where we will show that the hierarchy collapses at the Büchi level for ω -explorable automata.

4. DECIDABILITY AND COMPLEXITY OF THE EXPLORABILITY PROBLEM

In this section, we prove that the explorability problem is decidable and EXPTIME-complete for NFAs. We also exhibit an exponential upper-bound for deciding the explorability of $[0, 2]$ -automata.

We start by showing in Section 4.1 decidability of the explorability problem for NFAs using the results of [BDG⁺19] as a black box. This yields an algorithm in 2-EXPTIME. We give in Section 4.2 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 [BDG⁺19], 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 4.3, directly treating the more general case of Büchi automata.

4.1. 2-ExpTime algorithm via a black box reduction. Let us start by recalling the population control problem (PCP) of [BDG⁺19].

Definition 4.1 (*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 [BDG⁺19] to be EXPTIME-complete. We will present here a direct exponential reduction from the explorability problem to the PCP.

Theorem 4.2 (Direct reduction to the PCP). *The NFA explorability problem is decidable and in 2-EXPTIME.*

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 or priorities 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 4.3. *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 [BDG⁺19] as a black box on (\mathcal{B}, f) . Since \mathcal{B} is of exponential size compared to \mathcal{A} , this achieves the proof of Theorem 4.2.

4.2. ExpTime-hardness of NFA explorability.

Theorem 4.4. *The NFA explorability problem is EXPTIME-hard.*

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 [BDG⁺19] 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 2.2, 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 will define its accepting set as $F_{\mathcal{B}} = Q_{\mathcal{B}} \setminus \{f\}$.

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 $L(\mathcal{C}) = (\Sigma_{\mathcal{C}})^*$, and \mathcal{A} accepts whenever one of its components accepts, 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 thus the proof of Theorem 4.4.

Remark 4.5. Using standard arguments, it is straightforward to extend Theorem 4.4 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 4.1. This complicated winning condition is also the source of difficulty in the problem of recognizing HD parity automata.

4.3. EXPTIME algorithm for $[0,2]$ -explorability. The present work is an extended version of [HK23], where the following was proven:

Theorem 4.6. *The explorability problem can be solved in EXPTIME for Büchi automata (and all simpler conditions: NFA, safety, reachability).*

The algorithm was adapted from the EXPTIME algorithm for the PCP from [BDG⁺19]. We will use a variant of this first algorithm, and thus recall here the main ideas of the latter 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 [BDG⁺19] 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 analysed 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.

In the present extended version, we will perform some tweaks to this construction, in order to give an EXPTIME algorithm for deciding explorability of $[0, 2]$ -automata:

Theorem 4.7. *The explorability problem can be solved in EXPTIME for $[0, 2]$ -automata (and all simpler conditions: Büchi, co-Büchi, safety, reachability).*

Let us first give a general scheme of the proof, before going to a more formal and detailed description.

We start with a $[0, 2]$ -automaton \mathcal{A} , and want to decide whether it is explorable.

Our aim is to build a parity game G of size exponential in \mathcal{A} such that deciding whether \mathcal{A} is explorable amounts to deciding the winner of G . We want G to be an abstraction of the explorability game, where the number of tokens is not explicitly present. 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} recognizing $L(\mathcal{A})$, and incorporate it into the arena. The size of \mathcal{D} is exponential with respect to \mathcal{A} , and the number of priorities is polynomial.

We then follow [BDG⁺19] and build the *capacity game* augmented with \mathcal{D} . In this game, Spoiler plays a letter at each step, and Determiniser chooses a subgraph of the run-DAG of \mathcal{A} , that intuitively describes all the transitions taken by his tokens. The condition inherited from the capacity game is enforced as well, in order to force Determiniser to play a run-DAG compatible with a finite number of tokens. Such a DAG is 2-winning if there are infinitely many steps containing a 2-transitions. If it is not 2-winning, it may be unable to have a given token avoid seeing 1-transitions infinitely often, which means that no run through the support would be accepting. Therefore, Determiniser also has, at all time, a "challenger" that they want to prevent from seeing a 1-transition. If they fail at such a task infinitely often, the run is said to be 1-losing.

By combining all these ingredients, we show that we can as in [BDG⁺19] obtain a parity game of exponential size characterizing explorability of \mathcal{A} , yielding the wanted EXPTIME algorithm.

We also remark that, as in [BDG⁺19], this construction gives a doubly exponential upper bound on the number of tokens needed to witness explorability. Moreover, the proof from [BDG⁺19] that this is tight also stands here.

Let us now give a more detailed description of the construction.

We consider a non-deterministic $[0, 2]$ -automaton $\mathcal{A} = (\Sigma, Q, q_0^A, \Delta_{\mathcal{A}}, \text{rk})$, where rk is a function $Q \rightarrow [0, 2]$ giving the parity rank of each transition.

Let \mathcal{D} be a deterministic parity automaton for $L(\mathcal{A})$, that we can obtain via any standard EXPTIME algorithm. We will also make use of the capacity game from [BDG⁺19], in particular let \mathcal{T} be the deterministic parity automaton built in [BDG⁺19, Thm 4.5], that accepts a run-DAG if and only if infinitely many tokens are needed to instantiate it.³ The alphabet of this automaton consists in *transfer graphs* of \mathcal{A} , i.e. subsets of $Q \times Q$. Both \mathcal{D} and \mathcal{T} have an exponential size and a polynomial number of priorities with respect to the size of \mathcal{A} .

Definition 4.8 ($[0, 2]$ -capacity game). The $[0, 2]$ -capacity game is played in the arena $\mathcal{P}(Q) \times Q \times Q_{\mathcal{D}} \times Q_{\mathcal{T}}$, called $[0, 2]$ -*capacity arena*. It is played as follows by Determiniser and Spoiler.

- The starting position is $S_0 = (\{q_0^A\}, q_0^A, q_0^{\mathcal{D}}, q_0^{\mathcal{T}})$.
- At any given step with position $(B, q, q^{\mathcal{D}}, q^{\mathcal{T}})$ with $q \in B$, Spoiler chooses a letter $a \in \Sigma$, then Determiniser chooses a transfer graph $G \subseteq \Delta_{\mathcal{A}}(B, a)$, i.e. a subset of possible a -transitions starting from B , and a state q' such that $(q, a, q') \in B$.
- If (q, a, q') is of priority 1, then Determiniser can switch q' to any state in $\text{Im}(G)$. This event is recorded as an *elimination*. Else, q' does not change.
- The play then moves to the position $(\text{Im}(G), q', \delta_{\mathcal{D}}(q^{\mathcal{D}}, a), \delta_{\mathcal{T}}(q^{\mathcal{T}}, G))$. I.e. the set of tokens is updated to the image of G , the state of the challenger is updated to q' , and the states of \mathcal{D} and \mathcal{T} are updated deterministically.

A play can be represented by a sequence $(B_0, q_0, q_0^{\mathcal{D}}, q_0^{\mathcal{T}}) \xrightarrow{a_1, G_1, e_1} (B_1, q_1, q_1^{\mathcal{D}}, q_1^{\mathcal{T}}) \xrightarrow{a_2, G_2, e_2} \dots$, where e_i is a bit with value 1 if and only if an elimination took place at step i . The state q_i will be called the *challenger*.

We say that Spoiler wins the play if the run $q_0^{\mathcal{D}} q_1^{\mathcal{D}} q_2^{\mathcal{D}} \dots$ of \mathcal{D} is parity accepting, while only finitely many G_i contain 2-transitions (from $\Delta_{\mathcal{A}}$) and there is an infinite number of eliminations. Spoiler also wins if the run $q_0^{\mathcal{T}} q_1^{\mathcal{T}} q_2^{\mathcal{T}} \dots$ of \mathcal{T} is parity accepting, witnessing

³The notation \mathcal{T} stands for "tracking list automaton" as it is called in [BDG⁺19].

that the sequence $G_1G_2\dots$ of transfer graphs cannot be instantiated with finitely many tokens.

The following lemmas will allow us to show that the game functions as intended, i.e.:

- the capacity game still captures the finiteness of number of tokens in this extended construction.
- the elimination mechanism is a sound abstraction of the $[0, 2]$ -parity condition.

It thus remains to prove that if Spoiler wins the $[0, 2]$ -capacity game, if and only if he wins the k -explorability game for all $k \in \mathbb{N}$.

Let us begin by an observation that will allow us to prove this result:

Lemma 4.9. *The $[0, 2]$ -capacity game is finite-memory determined.*

Proof. The winning condition is a boolean condition of parity conditions, hence the game is ω -regular, and thus finite-memory determined. \square

In order to relate the $[0, 2]$ -capacity game to the k -explorability game, we will define the notion of projection of a play.

Definition 4.10 (Projection of a play). The *support arena* is the $\mathcal{P}(Q)$ component of the $[0, 2]$ -capacity arena, where Spoiler plays letters and Determiniser plays transfer graphs. Given a play $S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} S_2 \dots$ in the k -explorability game, the projection of that play in the support arena is the play $B_0 \xrightarrow{a_1, G_1} B_1 \xrightarrow{a_2, G_2} B_2 \dots$, where:

- B_i is the support of S_i (states occupied in S_i),
- $G_{i+1} = \{(S_i(j), S_{i+1}(j)) \mid j \in [0, k-1]\}$.

This corresponds to forgetting the multiplicity of tokens and only keeping track of the transitions that are used. Any play in the $[0, 2]$ -capacity game induces a play in the support arena as well, by simply forgetting the challenger and the deterministic components \mathcal{D} and \mathcal{T} .

In the following, we will not recall in detail the notion of capacity or other intricacies of the construction from [BDG⁺19], and will try to use them in a blackbox manner as much as possible. Since the $[0, 2]$ -capacity game is an extension of the capacity game from [BDG⁺19] with extra components, some results can be readily applied.

In this spirit, combining [BDG⁺19, Lem 3.5] with finite-memory determinacy of the $[0, 2]$ -capacity game, we obtain the following lemma:

Lemma 4.11. *If Determiniser has a finite-memory winning strategy τ in the $[0, 2]$ -capacity game, then he has a strategy σ in the k -tokens explorability game for some k , such that any play consistent with σ has its projection consistent with τ . Additionally, it is possible to choose σ such that the challenger token in the $[0, 2]$ -capacity game is always instantiated by a particular token in the k -explorability game, that can change at each elimination event in the $[0, 2]$ -capacity game.*

Proof. The strategy τ is in particular winning for the original capacity game, as Spoiler wins any play where \mathcal{T} accepts. We can therefore directly import the results from [BDG⁺19]: the finite memory m of τ ensures that the capacity of any winning play is actually bounded by a constant depending on m . From there, [BDG⁺19, Lem 3.5] ensures that there is a strategy that moves k tokens to realize the transfer graphs of τ , for some k exponential in m . This yields overall a doubly exponential upper bound on the number of tokens needed to

instantiate the strategy τ . Let us now describe how to build σ in order to additionally ensure that the challenger token is always instantiated by a particular token in the k -explorability game, that can only change at an elimination event. Whenever the challenger is instantiated (at the beginning or at each elimination), σ will simply choose the token of minimal index r among those in the new challenger state. Then, as all tokens play the same role in the instantiation strategy from [BDG⁺19], it is always possible to have token r follow the challenger path in σ , until the next elimination event or forever if there is no more elimination. Indeed, we are guaranteed that the wanted transition is always available, as it is part of the current transfer graph G . \square

We can now move to the main results of this section:

Lemma 4.12. *If Determiniser wins the $[0, 2]$ -capacity game, then he wins the k -explorability game for some $k \in \mathbb{N}$.*

Proof. Since the $[0, 2]$ -capacity game is finite-memory determined, we can assume that Determiniser has a finite-memory strategy τ , allowing us to apply Lemma 4.11. We will describe how to build a strategy for Determiniser in the k -explorability game, for some k given by Lemma 4.11. First of all, we have to lift the actual play in the k -explorability game to a play in the $[0, 2]$ -capacity game. This is done by simply projecting the set of tokens onto their support. The additional choices of challenger token will be given by the strategy τ . Determiniser is therefore able to simulate τ in this projected $[0, 2]$ -capacity, against letters played by Spoiler. We can then apply Lemma 4.11 to obtain a strategy σ in the k -explorability game, instantiating the behaviour of τ by actual tokens. Let us show that the resulting strategy σ from Lemma 4.11 is indeed winning in the k -explorability game. Let π be a play of the strategy τ in the $[0, 2]$ -capacity game, yielding a corresponding play π' of the strategy in the k -explorability game. If Determiniser wins π by witnessing infinitely many priorities 2, then one of the k tokens in π' will see infinitely many 2-transitions as well, and therefore π' is winning for Determiniser. Otherwise, if Determiniser wins π by preventing the challenger from seeing a 1-transition, then by the additional property requested of σ in Lemma 4.11, one of the tokens in π' will never see a 1-transition after some point, and therefore π' is winning for Determiniser as well. This achieves the proof that Determiniser has a winning strategy in the k -explorability game. \square

Let us now show the converse direction:

Lemma 4.13. *If Determiniser wins the k -explorability game for some $k \in \mathbb{N}$, then he wins the $[0, 2]$ -capacity game.*

Proof. Here we do not need any finite-memory determinacy result. Let σ be a winning strategy for Determiniser in the k -explorability game. We will show that this strategy can be lifted to a winning strategy τ in the $[0, 2]$ -capacity game. When playing in the $[0, 2]$ -capacity game, Determiniser will keep in memory a corresponding play of the k -explorability game, and follow the strategy σ in it. Then, to answer to the letters played by Spoiler, Determiniser will do the following:

- project the k tokens to their support, and play the corresponding transfer graph in the $[0, 2]$ -capacity game
- for the challenger, simply loop over all tokens: when the current challenger is token i and sees a 1-transition, an elimination event is witnessed, and the challenger is switched to token $i + 1$, looping back to 0 after $k - 1$.

Since the strategy σ is winning in the k -explorability game, the corresponding play in the $[0, 2]$ -capacity game will be winning as well:

- the automaton \mathcal{T} will be rejecting as the transfer graphs can be instantiated with a finite number of tokens
- Either infinitely many 2 will be seen in the run-DAG, or after some point the challenger will never see a 1-transition.

This achieves the description of a winning strategy τ for Determiniser in the $[0, 2]$ -capacity game. \square

To conclude and obtain 4.7, we have to show that the $[0, 2]$ -capacity game can actually be solved with the wanted complexity.

Theorem 4.14. *The $[0, 2]$ -capacity game can be solved in EXPTIME.*

Proof. Let \mathcal{G} be the $[0, 2]$ -capacity game associated to a $[0, 2]$ -automaton \mathcal{A} . We will show that the winning condition of the game \mathcal{G} for Spoiler can be seen as a disjunction of parity conditions. Recall that the winning condition for Spoiler is (\mathcal{T} accepts) or (\mathcal{D} accepts and finitely many 2 and infinitely many eliminations). Therefore, it is of the form $\text{Parity} \vee (\text{Parity} \wedge \text{coBüchi} \wedge \text{Büchi})$. But we can turn the second disjunct into a parity condition, at the price of some memory. Indeed, $\text{coBüchi} \wedge \text{Büchi}$ amounts to a $[1, 2, 3]$ condition. A condition of the form $[1, 2j]\text{-Parity} \wedge [1, 2, 3]\text{-Parity}$ can then be accepted by deterministic $[1, 2j + 1]$ -parity automaton \mathcal{D}' performing the following task:

- Whenever a 3-transition is seen on the second component, produce a rejecting $2j + 1$ rank,
- During sequence of 1-transitions on the second component, produce a rank of 1 and remember the highest rank h seen on the first component. This takes $2j$ states.
- When a 2-transition is seen on the second component, produce h .

This automaton \mathcal{D}' has a size polynomial in \mathcal{A} , as the number of priorities of \mathcal{D} is polynomial in \mathcal{A} . Thus, by incorporating \mathcal{D}' in the game, we obtain a game \mathcal{G}' equivalent to \mathcal{G} , still of size exponential in \mathcal{A} , and with winning condition a disjunction of two parity conditions. Such games are studied in [CHP07], which gives us an algorithm for solving \mathcal{G}' in time $O(m^{4d}m^2) \frac{(2d)!}{d!^2}$, where d is the number of priorities and m the size of the game.

If we take $n = |\mathcal{A}|$, using the fact that $m = O(2^{\text{poly}(n)})$ and $d = \text{poly}(n)$, we obtain an overall EXPTIME complexity for solving \mathcal{G} . \square

Remark 4.15. We can also be interested in the number of tokens needed for Determiniser to witness explorability of an automaton. By inspecting our proof, we can see that we obtain a doubly exponential upper bound. Moreover, we can use the same construction as in [BDG⁺19, Prop 6.3] to show that this is tight, *i.e.* some automata require a doubly exponential number of tokens to witness explorability. It is straightforward to lift this lower bound to the more difficult problem of NFA explorability (or more complex conditions on infinite words), so we do not detail this proof here.

4.4. The Parity explorability problem. We leave open the decidability of the explorability problem for parity automata beyond index $[0, 2]$.

However, we remark that from Lemma 3.6, the only remaining case to be treated is index $[1, 3]$. Indeed, for any parity automaton \mathcal{A} with n states and d parity ranks, Lemma

3.6 allows us to reduce (in polynomial time) explorability of \mathcal{A} to that of an equivalent $[1, 3]$ -automaton with $\frac{nd}{2}$ states.

5. EXPLORABILITY WITH COUNTABLY MANY TOKENS

In this section, we look at a variant of explorability where 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.

5.1. Definition and basic results. The following definition extends the notion of k -explorability to non-integer cardinals:

Definition 5.1 (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 5.2.

- *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 [Niw91] (even in the 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 5 (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 5. A winning strategy for Spoiler against ω 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. \square

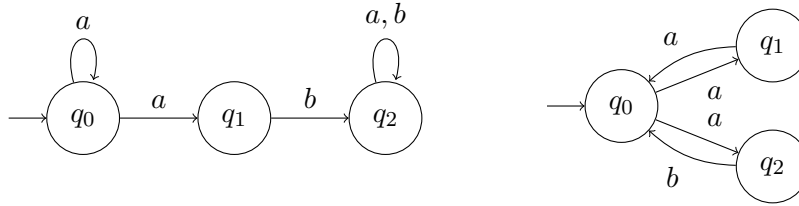


Figure 5: Two safety automata.

Left: ω -explorable but not explorable.

Right: not ω -explorable.

The first item of Lemma 5.2 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 5.3. *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 $w \in \Sigma^*$ labelling 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, and all accepted words are accepted after a finite run, 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 before the statement of the Lemma. Since acceptance in a reachability automaton is witnessed at a finite time, this strategy is winning. \square

5.2. ExpTime algorithm for coBüchi automata. We already know, from the example of Figure 5, that the result from Lemma 5.3 does not hold in the case of safety automata. For automata which are not automatically ω -explorable, we aim at deciding their ω -explorability status. We show the following decidability result on coBüchi automata. It holds in particular for safety automata as a subclass of coBüchi.

Theorem 5.4. *The ω -explorability of coBüchi 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 coBüchi (complete) automaton. We start by building a deterministic coBüchi automaton \mathcal{D} for $L(\mathcal{A})$ (e.g. using the breakpoint construction [MH84]). We will assume here that the coBüchi condition of \mathcal{A} is state-based to simplify a bit the presentation, but as remarked in the introduction it is straightforward to accomodate transition-based acceptance as well.

Definition 5.5 (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 the *challenger* q is not a coBüchi state, Protector picks a state $q' \in \Delta_{\mathcal{A}}(q, a)$.
- If the challenger q is a coBüchi state, Protector picks any state $q' \in \Delta_{\mathcal{A}}(B, a)$ as the new challenger. 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 $(B_0, q_0, p_0) \xrightarrow{a_1} (B_1, q_1, p_1) \xrightarrow{a_2} (B_2, q_2, p_2) \dots$, and Eliminator wins if infinitely many q_i and finitely many p_i are coBüchi states.

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

Notice that this is similar to the technique used for $[0, 2]$ -explorability, except that in absence of the capacity gadget \mathcal{T} , we do not enforce that finitely many tokens are used.

Lemma 5.6. *The elimination game is positionally determined and 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 coBüchi states p_i of \mathcal{D} with rank 3, the coBüchi states q_i of \mathcal{A} with rank 2, and the others with 1, then take the highest rank in (B_i, q_i, p_i) (ignoring B_i), Eliminator wins if and only if the highest rank appearing infinitely often is 2. As any parity game with 3 ranks can be solved in polynomial time [CJK⁺17], this is enough to get the result. Since parity games are positionally determined [EJ91], the elimination game is as well. \square

We want to prove the equivalence between this game and the ω -explorability game to obtain Theorem 5.4.

Lemma 5.7. *\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 $f : \mathbb{N} \rightarrow \mathbb{N}$

such that for any $n \in \mathbb{N}$, $|f^{-1}(n)|$ is infinite (for instance we can take f described by the sequence $0, 0, 1, 0, 1, 2, 0, 1, 2, 3, \dots$). The strategy for Spoiler will focus on sending token $f(0)$, then $f(1)$, then $f(2)$, *etc.* to a coBüchi state. Let σ be a memoryless winning strategy for Eliminator in the elimination game. 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 challenger is given by the last component via f : 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 coBüchi 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 challenger is the token $f(n)$ corresponds, on the exploration game, to that token visiting a coBüchi state. Ultimately this means that Determiniser did not provide any accepting run on any token (by definition of f that visits each index infinitely many times), while Spoiler did play a word from $\mathcal{L}(\mathcal{A})$, and therefore Spoiler wins.

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 challenger in the elimination game. When that token visits a coBüchi state, we switch to a token in the new challenger state 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 challenger token at that point never visits another coBüchi state, which also implies that Determiniser wins. \square

With Lemmas 5.6 and 5.7 we get a proof of Theorem 5.4, since the elimination game associated to \mathcal{A} is of exponential size and can be built using exponential time.

5.3. ExpTime-hardness of the ω -explorability problem.

Theorem 5.8. *The ω -explorability problem for (any automaton model embedding) safety automata is EXPTime-hard.*

Before giving the detailed proof of this result, we will give a quick sketch to convey the main ideas.

5.3.1. Proof sketch of Theorem 5.8.

We give a quick summary of the proof in this section. The full proof can be found in Section 5.3.2. 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 [BDG⁺19] showing EXPTime-hardness of the NFA population control problem (defined in Section 4.1).

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 [BDG⁺19]. 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 indicate which transition of the ATM he wants to follow.

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 \perp . He can then restart with the remaining tokens until all are in the target state \perp . 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 additionally lets Spoiler 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 5.8.

5.3.2. Detailed proof of Theorem 5.8.

We take an alternating Turing machine $\mathcal{M} = (\Sigma_{\mathcal{M}}, Q_{\mathcal{M}}, \Delta_{\mathcal{M}}, q_0^{\mathcal{M}}, q_f^{\mathcal{M}})$ with $Q_{\mathcal{M}} = Q_{\exists} \uplus Q_{\forall}$ and $\Delta_{\mathcal{M}} \subseteq Q_{\mathcal{M}} \times \Sigma_{\mathcal{M}} \times Q_{\mathcal{M}}$. It can be seen as a game between two players: existential (\exists) and universal (\forall). On a given input, the game creates a run by starting from $q_0^{\mathcal{M}}$, and letting \exists (resp. \forall) solve the non-determinism in states from Q_{\exists} (resp. Q_{\forall}) by picking a transition from $\Delta_{\mathcal{M}}$ compatible with the current letter of the input word. Player \exists wins if the play reaches the accepting state $q_f^{\mathcal{M}}$, and w is accepted if and only if \exists has a winning strategy on w . We assume that \mathcal{M} uses polynomial space $P(n)$ in the size n of its

input, *i.e.* the winning strategies can avoid configurations with tape longer than $P(n)$. We also fix an input word $w \in (\Sigma_{\mathcal{M}})^*$.

We will assume for simplicity that $\Sigma_{\mathcal{M}} = \{0, 1\}$ and that the machine alternates between existential and universal states, starting with an existential one (meaning that $q_0 \in Q_{\exists}$ and the transitions are either $Q_{\exists} \rightarrow Q_{\forall}$ or $Q_{\forall} \rightarrow Q_{\exists}$). In our reduction, this will mean that we give the choice of the transition alternatively to Spoiler (playing \exists) and Determiniser (\forall).

We create a safety automaton $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \perp)$ with:

- $Q = Q_{\mathcal{M}} \uplus Pos \uplus Mem \uplus Trans \uplus \{q_0, \text{store}, \top, \perp\}$ where:

$$\begin{aligned} Pos &= [1, P(n)] && \text{(representing where is the reading head on } \mathcal{M}'\text{s tape)} \\ Mem &= \{m_{b,i} \mid b \in \{0, 1\}, i \in [1, P(n)]\} && \text{(representing the current content of } \mathcal{M}'\text{s tape)} \\ Trans &= \{E\} \cup \{A_t \mid t \in \Delta_{\mathcal{M}}\} && \text{(encoding players' choices)} \end{aligned}$$

- $\Sigma = \{a_{t,p} \mid t \in \Delta_{\mathcal{M}} \text{ and } p \in [1, P(n)]\} \uplus \{\text{init}, \text{end}, \text{restart}, \text{win}\} \uplus \{\text{check}_q \mid q \in Q_{\mathcal{M}}\} \uplus \{\text{check}_{b,i} \mid (b, i) \in \{0, 1\} \times [1, P(n)]\}$.
- \perp is a rejecting sink state: a run is accepting if and only if it never reaches this state.

Let us give the intuition for the role of each state of \mathcal{A} . First, the states in $Q_{\mathcal{M}}$, Pos and Mem are used to keep track of the configuration of \mathcal{M} , as described in Lemma 5.9. Those in $Trans$ are used to simulate the choices of \exists and \forall (played by Spoiler and Determiniser respectively). The state **store** keeps tokens safe for the remaining of a run when Spoiler decides to ignore their transition choice. The sinks \top and \perp are respectively the one Spoiler must avoid at all cost, and the one in which he wants to send every token eventually.

We now define the transitions in Δ . The states \top and \perp are both sinks (\top accepting and \perp rejecting). We then describe all transitions labelled by the letter $a_{t,p}$ with $p \in Pos$ and $t = (q, q', b, b', d) \in \Delta_{\mathcal{M}}$, where q and q' are the starting and destination states of t , b and b' are the letters read and written at the current head position, and $d \in \{L, R\}$ is the direction taken by the head. These transitions are:

- $q \rightarrow q'$.
- $p \rightarrow p'$ with $p' = p + 1$ if $d = R$, or $p - 1$ if $d = L$. It goes to \top if $p' \notin [1, P(n)]$.
- $m_{b,p} \rightarrow m_{b',p}$, and $m_{b'',p''} \rightarrow m_{b'',p''}$ for any b'' and any $p'' \neq p$.
- $E \rightarrow A_{t'}$ for any transition t' .
- $A_t \rightarrow E$.
- $q'' \rightarrow \top$ for any $q'' \neq q$.
- $m_{\neg b,p} \rightarrow \top$ ($\neg b$ is the boolean negation of b).
- $p' \rightarrow \top$ for any $p' \neq p$.
- $A_{t'} \rightarrow \text{store}$ for any transitions $t' \neq t$.

This is represented in Figure 6.

The first three bullet points manage the evolution of the configuration of \mathcal{M} . The next two deal with the alternation between players, and the next three punish Spoiler if the transition is invalid (the **check** letters will handle the case where Determiniser is the one giving an invalid transition). The last one saves the tokens that are not chosen for the transition.

The other letters give the following transitions.

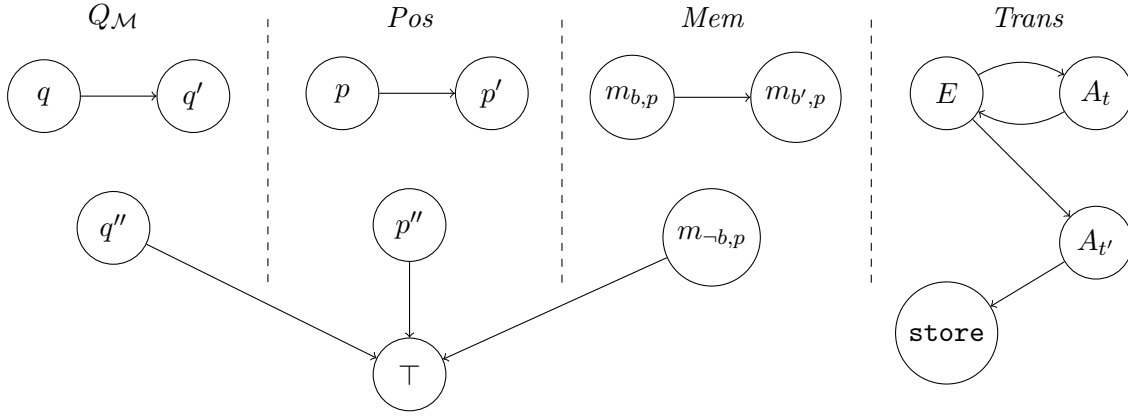


Figure 6: Transitions for $a_{t,p}$, where $t = (q, q', b, b', d)$, p' is the position at direction d from p , and q'', p'' , and t' are different from q, p, t respectively.

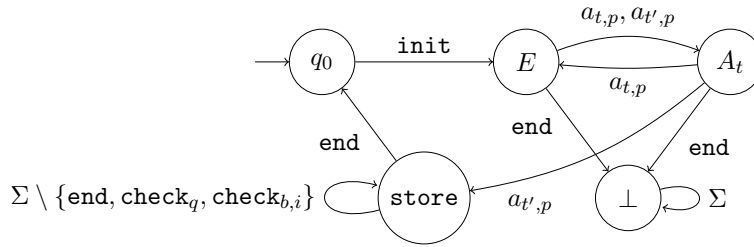


Figure 7: Gadget for simulating the choice of \forall in the alternation (transitions labelled by **check** are not represented, and t' represents any transition different from t).

- **init** goes from q_0 to the states E , $q_0^{\mathcal{M}}$, and $1 \in Pos$, and also to the states $m_{b,p}$ corresponding to the initial content of the tape, *i.e.* all $m_{b,p}$ such that b is the p -th letter of w (or 0 if $i > |w|$).
- **end** labels transitions from any non-accepting state of \mathcal{M} to \top , from **store** to q_0 , and from any other state to \perp .
- **check_q** creates a transition from A_t to \perp for any $t \in \Delta$ starting from q . It also creates a transition from q to \top . Any other state is sent back to q_0 . Intuitively, playing that letter means that q is not the current state and that any transition starting from q is invalid.
- **check_{b,p}** creates a transition from A_t to \perp for any $t \in \Delta$ reading b on the tape. It also creates transitions from any $j \in Pos \setminus \{p\}$ and from $m_{b,p}$ to \top . Any other state is sent to q_0 . Intuitively, playing that letter means that the current head position is p , and that its content is not b , so any transition reading b is invalid.

To summarize, the states of \mathcal{A} can be seen as two blocks, apart from q_0 , \top and \perp : those dealing with the configuration of \mathcal{M} ($Q_{\mathcal{M}}$, Pos and Mem), and those from the gadget of Figure 7 which deal with the alternation and non-deterministic choices.

The following result provides tools to manipulate the relation between \mathcal{A} and \mathcal{M} .

Lemma 5.9. *Let us consider a play of the ω -explorability game on \mathcal{A} , that we stop at some point. Suppose that the letters $a_{t,p}$ played since the last **init** are $a_{t_1,p_1}, \dots, a_{t_k,p_k}$. If \top is*

not reachable from q_0 with this sequence, then we can define a run ρ of \mathcal{M} on w taking the sequence of transitions t_1, \dots, t_k . The following implications hold:

Token present in	implies that at the end of ρ
$q \in Q_{\mathcal{M}}$	the current state is q
$p \in Pos$	the head is in position p
$m_{b,p} \in Mem$	the tape contains b at position p
E	it is the turn of \exists
A_t	it is the turn of \forall

Proof. These results are obtained by straightforward induction from the definitions. The unreachability of \top is used to ensure that only valid transitions are played. \square

We will now prove that \mathcal{A} is ω -explorable if and only if the Turing machine \mathcal{M} rejects the word w . Let us first assume that $w \in \mathcal{L}(\mathcal{M})$. There is a winning strategy σ_{\exists} for \exists in the alternating Turing machine game, and Spoiler will use that strategy in the explorability game to win against ω tokens. He will consider that the tokens are labelled by integers, and always target the smallest one that is not already in \perp . He proceeds as follows.

- Spoiler plays **init** from a position where every token is either in q_0 or \perp . We can assume from here that Determiniser sends tokens to each possible state, and just add imaginary tokens if he does not. Additionally, if the target token does not go to E , then it means that it is in a deterministic part of the automaton. In this case Spoiler creates an imaginary target token in E that will play only valid transitions (we will describe what this means later). Its purpose is to ensure that we actually reach an accepting state of \mathcal{M} to destroy the real target token.
- When there are tokens in E , Spoiler plays letters according to σ_{\exists} . More formally, if the letters played since **init** are $a_{t_1, p_1} \dots a_{t_i, p_i}$, then Spoiler plays $a_{t_{i+1}, p_{i+1}}$ where $t_{i+1} = \sigma_{\exists}(t_1, \dots, t_i)$ and $p_{i+1} = p_i + 1$ or $p_i - 1$ depending on the head movement in t_i .
- After such a play, Determiniser can move tokens to any state A_t . If there is more than one occupied state, Spoiler picks the one containing the current target token (possibly imaginary).
 - If that state corresponds to an invalid transition (wrong starting state or wrong tape content at the current head position), then Spoiler plays the corresponding **check** letter. Formally, if the target token (not the imaginary one, since Spoiler can avoid invalid transitions for that one) is in A_t , Spoiler plays **check** $_q$ if the starting state q of t does not match the current state of the tape (given by Lemma 5.9), or **check** $_{b,p}$ if the current head position is p and does not contain b . In both cases, the target token is sent to \perp with no other token reaching \top (by Lemma 5.9). This sends us back to the first step, but with an updated target.
 - If the state instead corresponds to a valid transition, then Spoiler can play the corresponding $a_{t,p}$, where p is the current head position (again, given by Lemma 5.9), then go back to the previous step (where there are tokens in E).
- If no invalid transition is reached, the run eventually gets to an accepting state of \mathcal{M} because σ_{\exists} is winning. This corresponds to a stage where Spoiler can safely play **end** to get rid of the target token along with all tokens outside of **store**, by sending them to \perp (the only reason not to play **end** would be the existence of tokens in non-accepting states of $Q_{\mathcal{M}}$). This sends us back to the first step, but with an updated target. Notice that if there was a virtual target token, we will always reach this event, and send the real target token (located in $Q_{\mathcal{M}}$ or Pos or Mem) in \perp .

This strategy guarantees that after k runs, at least the first k tokens are in state \perp , and therefore cannot witness an accepting run. We also know that the final word is accepted by \mathcal{A} , because an accepting run can be created by going to the state `store` as soon as possible in each factor corresponding to a run of \mathcal{M} .

Conversely, if there is a winning strategy σ_{\forall} for the universal player in the alternation game on $\mathcal{M}(w)$, then we can build a winning strategy for Determiniser in the ω -explorability game. This strategy is more straightforward than the previous one, as we can focus on the tokens sent to E (while still populating each state when `init` is played, but these other tokens follow a deterministic path until the next `init`).

Determiniser will initially choose a specific token, called leader. He then sends ω tokens to every reachable state when Spoiler plays `init`, with the leader going to E . Determiniser then moves the tokens in the leader's state according to σ_{\forall} . Spoiler cannot send the leader to \perp , since the only way to do that would be using the letter `end`, but this would immediately ensure the win for Determiniser, as there will always be some token in non-accepting states of \mathcal{M} (because σ_{\forall} is winning), and those tokens would be sent to \top upon playing `end`. This means that Spoiler has no way to send the leader to \perp without losing the game, and therefore Determiniser wins.

Note that with that strategy, Spoiler can still safely send some tokens to \perp by playing the wrong transition, which sends the tokens following the leader to `store`, then some well-chosen `check` letter to send the remaining ones to \perp . However, Determiniser will start the next run with still ω tokens, including the leader. This is why the choice of a specific leader is important, as it can never be safely sent to \perp .

This proves that the automaton \mathcal{A} created from \mathcal{M} and w (using polynomial time) is ω -explorable if and only if \mathcal{M} rejects w . This completes the proof, since the acceptance problem is EXPTIME-hard for alternating Turing machines using polynomial space.

5.4. Büchi case, or the general case. Surprisingly, compared to the situation with a finite number of tokens, in the ω -explorability case, the expressivity hierarchy collapses as early as the Büchi case, as ω -explorable Büchi automata can recognize all ω -regular languages. We can even build in PTIME a Büchi automaton whose ω -explorability is equivalent to the one of an input parity automaton.

Theorem 5.10. *Let \mathcal{A} be a parity automaton. We can build in PTIME a Büchi automaton \mathcal{B} recognizing $\mathcal{L}(\mathcal{A})$, such that \mathcal{B} is ω -explorable if and only if \mathcal{A} is ω -explorable.*

Proof. Intuitively, the idea is to build an automaton that will make a case disjunction over the different even parities l of \mathcal{A} , and will ensure that the run never encounters any priority $> l$ after some time. This is a classical way to turn a parity automaton into a Büchi one. We just need to ensure that the non-determinism introduced in this construction preserve ω -explorability.

Let us first describe formally the construction. We define, for l even parity of \mathcal{A} , a copy \mathcal{A}_l of \mathcal{A} where all transitions of priority $< l$ become non-Büchi transitions, all transitions of priority l become Büchi transitions, and all transitions of priority $> l$ are rerouted towards a rejecting sink state \perp . The automaton \mathcal{A}_l recognizes the language of words of $\mathcal{L}(\mathcal{A})$ which can be accepted in \mathcal{A} with infinitely many priorities l and never encounter any priority $> l$. We define \mathcal{A}' as the copy of \mathcal{A} where the rank of all transitions is changed to 1 (i.e. non-Büchi).

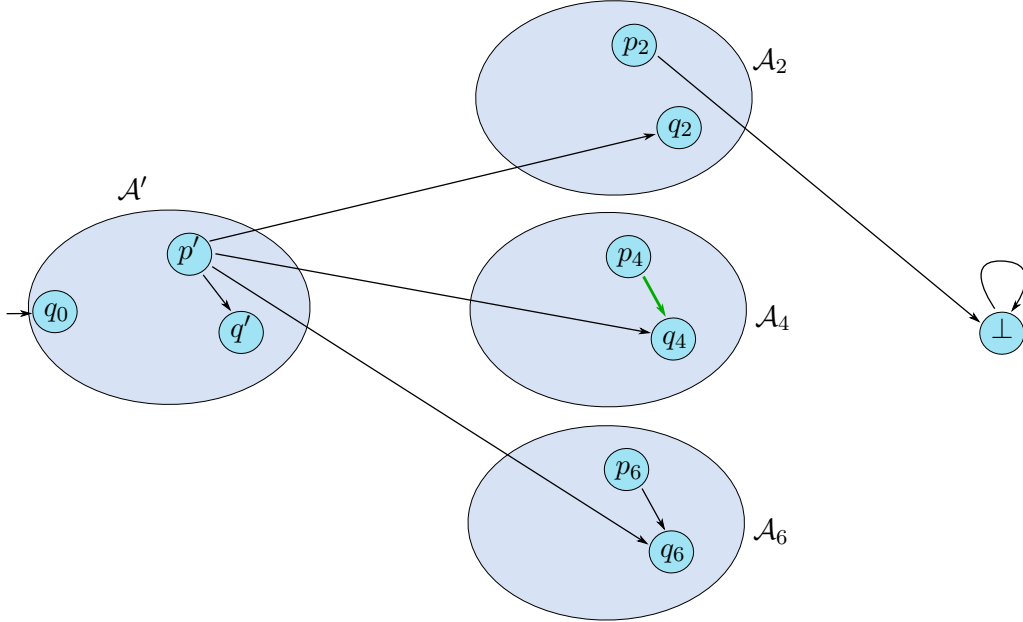


Figure 8: Illustration of the construction of \mathcal{B} from \mathcal{A} for the proof of Theorem 5.10. All transitions associated to a transition $p \xrightarrow{a:4} q$ are represented (label a omitted), in bold green for the Büchi transition.

The automaton \mathcal{B} will simply be the union of \mathcal{A}' all the \mathcal{A}_l : the run starts in \mathcal{A}' , then can non-deterministically jump to any \mathcal{A}_l at any time, keeping the local state coherent. The transitions of \mathcal{B} are those of \mathcal{A}' , \mathcal{A}_l , plus transition of the form $p' \xrightarrow{a} q_l$ with $p' \in \mathcal{A}'$ and $q_l \in \mathcal{A}_l$ for some l , corresponding to a transition $p \xrightarrow{a} q$ in the original automaton \mathcal{A} , regardless of priorities. See Figure 8 for an illustration.

It is clear that $L(\mathcal{B}) = L(\mathcal{A})$: an accepting run in \mathcal{B} must jump to some \mathcal{A}_l at some point, and from there witness that the word is accepted in \mathcal{A} with priority l . Conversely, a l -accepting run in \mathcal{A} can be simulated in \mathcal{B} by jumping to the corresponding \mathcal{A}_l after the last priority $> l$ is encountered.

We will now show that \mathcal{B} is ω -explorable if and only if \mathcal{A} is ω -explorable.

\implies If \mathcal{B} is ω -explorable, the strategy for Determinizer can simply be copied to \mathcal{A} , by projecting states and transitions of \mathcal{B} to \mathcal{A} in the canonical way. When a token follows an accepting run in \mathcal{B} , the corresponding token will follow an accepting run in \mathcal{A} , so this strategy witnesses ω -explorability of \mathcal{A} .

\impliedby If \mathcal{A} is ω -explorable with σ winning strategy for Determinizer, we will build a winning strategy σ' for Determinizer in the ω -explorability game \mathcal{B} .

To do so, we will associate to each token i of σ a countable set of tokens $\{t_{i,j,l} \mid j \in \mathbb{N}, l \text{ even parity of } \mathcal{A}\}$. The strategy σ' will have token $t_{i,j,l}$ will follow the same path as token i in σ , starting out in copy \mathcal{A}' , and jumping in copy \mathcal{A}_l at time j .

Since some token i accepts in σ , by seeing infinitely many priority l , with no priority $> l$ after some time j , the token $t_{i,j,l}$ will accept according to σ' .

There are still countably many tokens (which can be re-indexed by \mathbb{N}), so this witnesses ω -explorability of \mathcal{B} . □

Theorem 5.10 gives us the following corollary:

Corollary 5.11. *If ω -explorability is decidable for Büchi automata, then it is decidable for parity automata.*

We leave open the decidability of ω -explorability for Büchi automata.

The expressivity picture is complete for ω -explorable automata: the hierarchy collapses at the Büchi level, while the coBüchi level recognizes only deterministic coBüchi languages, as it is the case for non-deterministic automata in general.

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 (and in particular decidable) for $[0, 2]$ -parity condition in the first case and coBüchi condition in the second case.

We leave open the cases of deciding explorability of $[1, 3]$ -automata and ω -explorability of Büchi automata. These correspond to the general case: (ω)-explorability of any parity automaton can be reduced to these cases.

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 [BL22] 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 non-determinism, and this notion could be used in other contexts as a middle ground between deterministic and non-deterministic automata. We also saw that despite its apparent abstractness, ω -explorability captures a natural property that we believe can be useful in verification: the ability of Spoiler to “kill” any run of its choice.

Acknowledgments. We thank Milla Valnet for her preliminary work on the explorability question.

REFERENCES

- [AJP24] Rohan Acharya, Marcin Jurdzinski, and Aditya Prakash. Lookahead games and efficient determination of history-deterministic büchi automata. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPICs*, pages 124:1–124:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [AK22] Bader Abu Radi and Orna Kupferman. Minimization and Canonization of GFG Transition-Based Automata. *Logical Methods in Computer Science*, Volume 18, Issue 3, August 2022. URL: <https://lmcs.episciences.org/9880>, doi:10.46298/lmcs-18(3:16)2022.
- [AKL21] 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.

- [BDG⁺19] Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, Hugo Gimbert, and Adwait Amit Godbole. Controlling a population. *Log. Methods Comput. Sci.*, 15(3), 2019.
- [BK18] 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.
- [BKKS13] 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.
- [BKLS20] 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.
- [BL21] 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.
- [BL22] 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.
- [Bok22] 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.
- [Cas23] Antonio Casares. *Structural properties of automata over infinite words and memory for games. (Propriétés structurelles des automates sur les mots infinis et mémoire pour les jeux)*. PhD thesis, University of Bordeaux, France, 2023.
- [CHP07] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized parity games. In Helmut Seidl, editor, *Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007*, Lecture Notes in Computer Science. Springer, 2007.
- [CJK⁺17] 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.
- [Col09] 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.
- [Col12] 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.
- [EJ91] 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.
- [GTW02] 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.
- [HK23] Emile Hazard and Denis Kuperberg. Explorable Automata. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*, volume 252 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CSL.2023.24.
- [HKK⁺00] Juraj Hromkovic, Juhani Karhumäki, Hartmut Klauck, Georg Schnitger, and Sebastian Seibert. Measures of nondeterminism in finite automata. *Electronic Colloquium on Computational Complexity (ECCC)*, 7, 01 2000.
- [HP06] Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *Computer Science Logic, 20th International Workshop, CSL 2006*, 2006.
- [KM19] Denis Kuperberg and Anirban Majumdar. Computing the width of non-deterministic automata. *Log. Methods Comput. Sci. (LMCS)*, 15(4), 2019.

- [KS15] 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.
- [McN93] Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993. doi:[https://doi.org/10.1016/0168-0072\(93\)90036-D](https://doi.org/10.1016/0168-0072(93)90036-D).
- [MH84] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω -words. *Theoret. Comput. Sci.*, 32(3):321–330, 1984.
- [Niw91] 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.
- [PSA17] 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.
- [WS91] Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theor. Comput. Sci.*, 88(2):325–349, 1991.