

Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games

Wojciech Czerwiński¹, Laure Daviaud², Nathanaël Fijalkow^{3,4,2}, Marcin Jurdziński²,
Ranko Lazić², and Paweł Parys¹

¹MIMUW, University of Warsaw, Poland

²DIMAP, Department of Computer Science, University of Warwick, UK

³CNRS, LaBRI, France

⁴The Alan Turing Institute, UK

Abstract

Several distinct techniques have been proposed to design quasi-polynomial algorithms for solving parity games since the breakthrough result of Calude, Jain, Khoussainov, Li, and Stephan (2017): *play summaries*, *progress measures* and *register games*. We argue that all those techniques can be viewed as instances of the *separation approach* to solving parity games, a key technical component of which is constructing (explicitly or implicitly) an automaton that *separates* languages of words encoding plays that are (decisively) won by either of the two players. Our main technical result is a quasi-polynomial lower bound on the size of such separating automata that nearly matches the current best upper bounds. This forms a *barrier* that all existing approaches must overcome in the ongoing quest for a polynomial-time algorithm for solving parity games. The key and fundamental concept that we introduce and study is a *universal ordered tree*. The technical highlights are a quasi-polynomial lower bound on the size of universal ordered trees and a proof that every *separating safety automaton* has a universal tree hidden in its state space.

1 Introduction

1.1 Parity games. A *parity game* is played on a directed graph by two players who are called Even and Odd. A play starts at a designated vertex and then the players move by following outgoing edges forever, thus forming an infinite path. Every vertex of the graph is owned by one of the two players and it is always the owner of the vertex who moves by following an outgoing edge from the current vertex to the next one.

This completes the description of the dynamics of a play, but how do we declare the winner of an infinite

path formed in this way? For this, we need to inspect positive integers that label all edges in the graph, which we refer to as *edge priorities*, or simply priorities. Player Even is declared the winner of a play if the highest priority that occurs infinitely many times is even, and otherwise player Odd wins; equivalently, the winner is the parity of the limesup (limes superior) of the priorities that occur in the play.

The principal algorithmic problem studied in the context of parity games is *deciding the winner*: given a game graph as described above and a starting vertex, does player Even have a winning strategy—a recipe for winning every play starting from the designated vertex, no matter what edges her opponent Odd follows whenever it is his turn to move.

Determinacy and complexity. A *positional strategy* for Even is a set of edges that go out of vertices she owns—exactly one such edge for each of her vertices; Even uses such a strategy by always—if the current vertex is owned by her—following the unique outgoing edge that is in the strategy. Note that when Even uses a positional strategy, her moves depend only on the current vertex—they are oblivious to what choices were made by the players so far. A basic result for parity games that has notable implications is their *positional determinacy* [12, 28]: for every starting vertex, exactly one of the players has a winning strategy and hence the set of vertices is partitioned into the *winning set* for Even and the winning set for Odd; moreover, each player has a *positional strategy* that is winning for her from all starting vertices in her winning set.

An important corollary of positional determinacy is that deciding the winner in parity games is *well characterized*, i.e., it is both in NP and in co-NP [13].

Several further complexity results suggest that it may be difficult to provide compelling evidence for hardness of solving parity games: deciding the winner is known to be also in UP and in co-UP [23], and computing winning strategies is in PLS, PPAD, and even in their subclass CLS [9, 10]. Parity games share this intriguing complexity-theoretic status with several other related problems, such as mean-payoff games [33], discounted games, and simple stochastic games [8], but they are no harder than them since there are polynomial reductions from parity games to mean-payoff games, to discounted games, and to simple stochastic games [23, 33].

Significance and impact. Parity games play a fundamental role in automata theory, logic, and their applications to verification and synthesis. Specifically, the algorithmic problem of deciding the winner in parity games is polynomial-time equivalent to the model checking in the modal μ -calculus and to checking emptiness of automata on infinite trees with parity acceptance conditions [13], and it is at the heart of algorithmic solutions to the Church’s synthesis problem [29].

The impact of parity games goes well beyond their place of origin in automata theory and logic. We illustrate it by the resolutions of two long-standing open problems in *stochastic planning* and in *linear programming*, respectively, that were directly enabled by the ingenious examples of parity games given by Friedmann [18], on which the *strategy improvement algorithm* [32] requires exponentially many iterations. Firstly, Fearnley [15] has shown that Friedmann’s examples can be adapted to prove that *Howard’s policy iteration* algorithm for Markov decision processes (MDPs) requires exponentially many iterations. Policy iteration has been well-known and widely used in stochastic planning and AI since 1960’s, and it has been celebrated for its fast termination: until Fearnley’s surprise result, no examples were known for which a superlinear number of iterations was necessary. Secondly, Friedmann, Hansen, and Zwick [19] have adapted the insights from the lower bounds for parity games and MDPs to prove that natural *randomized pivoting rules* in the *simplex algorithm* for linear programming may require subexponentially many iterations. The following quote from the full version of Friedmann et al. [19] highlights the role that parity games (PGs) played in their breakthrough:

“our construction can be described and understood without knowing about PGs. We would like to stress, however, that most of our intuition about the problem was obtained by thinking in terms of PGs. Thinking in terms of MDPs seems harder, and we doubt whether we could have obtained our results by thinking

directly in terms of linear programs.”

In both cases, Friedmann’s examples of parity games and their analysis have been pivotal in resolving the theoretical worst-case complexity of influential algorithms that for many decades resisted rigorous analysis while performing outstandingly well in practice.

Current state-of-the-art. It is a long-standing open question whether there is a polynomial-time algorithm for solving parity games [13]. The study of algorithms for solving parity games has been dominated for over two decades by algorithms whose run-time was exponential in the number of distinct priorities [14, 4, 31, 24, 32, 30], or mildly subexponential for large number of priorities [2, 26]. The breakthrough came in 2017 from Calude et al. [5] who gave the first quasi-polynomial-time algorithm using the novel idea of *play summaries*. Several other quasi-polynomial-time algorithms were developed soon after, including space-efficient *progress-measure* based algorithms of Jurdziński and Lazić [25] and of Fearnley, Jain, Schewe, Stephan, and Wojtczak [16], and the algorithm of Lehtinen [27], based on her concept of *register games*.

1.2 The separation approach. Bojańczyk and Czerwiński [3, Section 3] have observed that the main technical contribution of Calude et al. [5] can be elegantly phrased using concepts from automata theory. They have pointed out that in order to reduce solving a parity game of size at most n to solving a conceptually and algorithmically much simpler *safety game*, it suffices to provide a finite *safety automaton* that achieves the task of *separating* two sets EvenLoops_n and OddLoops_n of infinite words that describe plays on graphs of size at most n that are *decisively won* by the respective two players. For encoding plays in parity games, they use words in which every letter is a pair that consists of a vertex and a priority. The definition of such a word being decisively won by a player that was proposed by Bojańczyk and Czerwiński is that the biggest priority that occurs on every cycle—an infix in which the first vertex and the vertex immediately following the infix coincide—is of her parity. Concerning separation, for two disjoint languages J and K , we say that a language S *separates* J from K if $J \subseteq S$ and $S \cap K = \emptyset$, and we say that an automaton \mathcal{A} is a *separator* of two languages if the language $L(\mathcal{A})$ of words recognized by \mathcal{A} separates them. The main technical contribution of Calude et al. [5] can then be stated as constructing separators—of quasi-polynomial size—of the languages EvenLoops_n and OddLoops_n .

Note that a separator of EvenLoops_n and OddLoops_n has a significantly easier task than a *rec-*

ognizer of exactly the set LimsupEven of words that are won by Even—that is required to accept all words in LimsupEven , and to reject all words in LimsupOdd , the set of all words that are won by Odd. Instead, a separator may reject some words won by Even and accept some words won by Odd, as long as it accepts all words that are decisively won by Even, and it rejects all words that are decisively won by Odd.

What Calude et al. [5] exploit is that if one of the players uses a *positional* winning strategy then all plays are indeed encoded by words that are won decisively by her, no matter how the opponent responds. The formalization of Bojańczyk and Czerwiński [3] is that—using positional determinacy of parity games [12, 28]—in order to solve a parity game of size at most n , it suffices to solve a strategically and algorithmically much simpler *safety game* that is obtained as a simple chained product of the parity game and a safety automaton that is a separator of EvenLoops_n and OddLoops_n .

1.3 Our contribution. Our main conceptual contributions include making explicit the notion of a *universal ordered tree* and unifying all the existing quasi-polynomial algorithms for parity games [5, 25, 20, 16, 27] as instances of the *separation approach* proposed by Bojańczyk and Czerwiński [3].

We point out that it is exactly the universality property of an ordered tree that makes it suitable for serving as a witness search space in *progress measure lifting* algorithms [24, 1, 30, 25, 11], and that the running time of such algorithms is dictated by the size of the universal tree used. In particular, by proving a quasi-polynomial lower bound on the size of universal trees in Section 2.4, we rule out the hope for improving progress measure lifting algorithms to work in sub-quasi-polynomial time by finding smaller universal trees. As our other main technical results in Section 5 show, however, universal trees are fundamental not only for progress measure lifting algorithms, but for all algorithms that follow the separation approach.

We argue that in the separation approach, it is appropriate to slightly adjust the choice of languages to be separated, from EvenLoops_n and OddLoops_n proposed by Bojańczyk and Czerwiński [3] to the more suitable EvenCycles_n and OddCycles_n (see Section 3.1 for the definitions). We also verify, in Section 4, that all the three distinct techniques of solving parity games in quasi-polynomial time considered in the recent literature (*play summaries* [5, 20, 16], *progress measures* [25], and *register games* [27]) yield separators for languages EvenCycles_n and LimsupOdd , which (as we argue in Section 3.2) makes them suitable for the separation approach.

The main technical contribution of the paper, described in Sections 2.4 and 5 is a proof that every (non-deterministic) safety automaton that separates EvenCycles_n from LimsupOdd has a number of states that is at least quasi-polynomial. Recall that in Section 2.4 we establish a quasi-polynomial lower bound on the size of universal trees. Then, in Section 5, our argument is based on proving that in every separating automaton as above, one can define a sequence of *linear quasi-orders* on the set of states, in which each quasi-order is a refinement of the quasi-order that follows it in the sequence. Such a sequence of linear quasi-orders can be naturally interpreted as an *ordered tree* in which every leaf is populated by at least one state of the automaton. We then also prove that the ordered tree must contain a *universal ordered tree*, and the main result follows from the earlier quasi-polynomial lower bound for universal trees.

Another technical highlight, presented in Section 4.1, is a construction of a separator from an arbitrary universal tree, which together with the lower bound in Section 5 implies that the sizes of smallest universal trees and of smallest separators coincide. The correctness of the construction relies on existence of *progress measures* that map from vertices of game graphs into leaves of universal trees, and that witness winning strategies.

The significance of our main technical results is that they provide evidence against the hope that any of the existing technical approaches to developing quasi-polynomial algorithms for solving parity games [5, 25, 16, 27] may lead to further improvements to sub-quasi-polynomial algorithms. In other words, our quasi-polynomial lower bounds for universal trees and separators form a *barrier* that all existing approaches must overcome in the ongoing quest for a polynomial-time algorithm for solving parity games.

2 Progress measures and universal trees

Progress measures are witnesses of winning strategies that underpin the design of the *small progress measure lifting algorithm* [24]. For nearly two decades, this algorithm and its variants [1, 30, 6, 25, 11] have been consistently matching or beating the worst-case performance guarantees of the state-of-the-art algorithms for solving parity games.

In this section we introduce the notion of *universal ordered trees* and we point out how the universality property uniformly justifies the correctness of progress measure lifting algorithms, and that the size of universal trees used in such algorithms drives their worst-case run-time complexity. Those observations motivate the key technical question that we tackle in

this section, namely whether the recent construction of quasi-polynomial universal trees by Jurdziński and Lazić [25] can be significantly improved to yield a sub-quasi-polynomial algorithms for solving parity games.

The main original technical contribution of this section is a negative answer to this question: we establish a quasi-polynomial lower bound on the size of universal trees that nearly matches (up to a small polynomial factor) the upper bound of Jurdziński and Lazić. This dashes the hope of obtaining a significantly faster algorithm for solving parity games by constructing smaller universal trees and running the progress measure algorithm on them.

2.1 Game graphs and strategy subgraphs.

Throughout the paper, we write V for the set of vertices and E for the set of edges in a parity game graph, and we use n to denote the numbers of vertices. For every edge $e \in E$, its *priority* $\pi(e)$ is a positive integer, and we use d to denote the smallest even number that priority of no edge exceeds. Without loss of generality, we assume that every vertex has at least one outgoing edge. We say that a cycle in a game graph is *even* if the largest edge priority that occurs on it is even; otherwise it is *odd*.

Recall that a positional strategy for Even is a set of edges that go out of vertices she owns—exactly one such edge for each of her vertices. The *strategy subgraph* of a positional strategy for Even is the subgraph of the game graph that includes all outgoing edges from vertices owned by Odd and exactly those outgoing edges from vertices owned by Even that are in the positional strategy. Observe that the set of plays that arise from Even playing her positional strategy is exactly the set of all plays in the strategy subgraph. Moreover, note that every cycle in the strategy subgraph of a positional strategy for Even that is winning for her is even: otherwise, by repeating an odd cycle indefinitely, we would get a play that is winning for Odd. Further overloading terminology, we say that a (parity game) graph is *even* if all cycles in it are even.

2.2 Ordered trees and progress measures.

An *ordered tree* is a prefix-closed set of sequences of a linearly ordered set. We refer to such sequences as *tree nodes*, we call the elements of such sequences *branching directions*, and we use the standard ancestor-descendent terminology for nodes. For example: node $\langle \rangle$ is the *root* of a tree; node $\langle x \rangle$ is the *child* of the root that is reached from the root via the branching direction x ; node $\langle x, y \rangle$ is the *parent* of node $\langle x, y, z \rangle$; nodes $\langle \rangle$, $\langle x \rangle$, and $\langle x, y \rangle$ are *ancestors* of node $\langle x, y, z \rangle$; and nodes $\langle x, y \rangle$ and $\langle x, y, z \rangle$ are *descendants* of nodes $\langle \rangle$ and $\langle x \rangle$. Moreover,

a node is a *leaf* if it does not have any descendants. All nodes in an ordered tree are linearly ordered by the *lexicographic order* on sequences that is induced by the assumed linear order on the set of branching directions. For example, we have $\langle x \rangle < \langle x, y \rangle$, and $\langle x, y, w \rangle < \langle x, z \rangle$ if $y < z$. The *depth* of a node is the number of elements in the eponymous sequence, the *height* of a tree is the maximum depth of a node in it, and the *size* of a tree is the number of its leaves.

A *tree labelling* of a parity game is a mapping μ from the vertices in the game graph to leaves in an ordered tree of height $d/2$; for convenience and without loss of generality we assume that every leaf has depth $d/2$. We write $\langle m_{d-1}, m_{d-3}, \dots, m_1 \rangle$ to denote such a leaf, and for every priority p , $1 \leq p \leq d$, we define its p -truncation $\langle m_{d-1}, m_{d-3}, \dots, m_1 \rangle|_p$ to be the sequence $\langle m_{d-1}, m_{d-3}, \dots, m_p \rangle$ if p is odd, and $\langle m_{d-1}, m_{d-3}, \dots, m_{p+1} \rangle$ if p is even. We say that a tree labelling μ of the game is a *progress measure* if the following *progress condition* holds for every edge (v, u) in the strategy subgraph of some positional strategy for Even:

- if p is even then $\mu(v)|_{\pi(v,u)} \geq \mu(u)|_{\pi(v,u)}$;
- if p is odd then $\mu(v)|_{\pi(v,u)} > \mu(u)|_{\pi(v,u)}$.

We recommend inspecting the (brief and elementary) proof of [25, Lemma 2], which establishes that every cycle in the strategy subgraph whose all edges satisfy the progress condition is even. It gives a quick insight into the fundamental properties of progress measures and it shows the easy implication in the following theorem that establishes progress measures as witnesses of winning strategies in parity games.

THEOREM 2.1. ([12, 24]) *Even has a winning strategy from every vertex in a parity game if and only if there is a progress measure on the game graph.*

2.3 Finding tree witnesses on universal trees.

It is a straightforward but fruitful observation of Jurdziński and Lazić [25] that a progress measure on a game graph with n vertices and at most d distinct edge priorities is a mapping from the vertices in the game graph to nodes in an ordered tree of height at most $d/2$ and *with at most n leaves* (all subtrees that no vertex is mapped to can be pruned). It motivates the fundamental concept that we introduce in this section—*universal trees*.

An (ℓ, h) -*universal (ordered) tree* is an ordered tree, such that every ordered tree of height at most h and with at most ℓ leaves can be isomorphically embedded into it; in such an embedding, the root of the tree must be mapped onto the root of the universal tree, and the children of every node must be mapped—injectively and in an order-preserving way—onto the children of its

image.

The following proposition follows directly from the above “straightforward but fruitful” observation and the definition of a universal tree.

PROPOSITION 2.1. ([25]) *Every progress measure on a graph with n vertices and with at most d priorities can be seen as a map into an $(n, d/2)$ -universal tree.*

We offer the following interpretation of the *small progress measure lifting* [24] and the *succinct progress measure lifting* [25] algorithms, highlighting the central role played in them by the concept of universal trees that we introduce here. Both algorithms perform an iterative search for a progress measure (a witness for a winning strategy for Even), and their search spaces are limited by restricting the labels considered in candidate tree labellings to leaves in specific $(n, d/2)$ -universal trees. Where the two algorithms differ is the universal trees they use: in the former algorithm it is the full n -ary tree of height $d/2$, and in the latter it is an ordered tree of height $d/2$ in which the branching directions are bit strings, where for every node, the total number of bits used in all its branching directions is bounded by $\lceil \lg n \rceil$.

We observe that what is common for both algorithms is that their correctness relies precisely on the universality property of the ordered tree from which the candidate labels are taken from. Indeed, if there is a witness for a winning strategy for Even in the form of a progress measure (whose existence is guaranteed by Theorem 2.1), then by Proposition 2.1, it suffices to look for one that uses only leaves of a universal tree as labels. The original papers [24, 25] contain the technical details of the iterative scheme that both algorithms use. Here, we highlight the following two key insights about the design and analysis of those algorithms:

- each algorithm computes a sequence of tree labellings that is monotonically increasing (w.r.t. the pointwise lexicographic order), while maintaining the following invariant: the current labelling is pointwise lexicographically smaller than or equal to every progress measure;
- the worst-case run-time bound for the algorithm is determined (up to a small polynomial factor) by the size of the universal tree used.

Using our interpretation of the progress measure lifting algorithms and the terminology of universal trees, the small progress measure lifting algorithm [24] is using a tree whose universality is straightforward, and whose size— $\Theta(n^{d/2})$ —is exponential in the number of priorities. On the other hand, the main technical contribution of Jurdziński and Lazić—that yields their quasi-polynomial succinct progress measure lifting algorithm [25, Theorem 7]—can be stated as the following

quasi-polynomial upper bound on the size of universal trees.

THEOREM 2.2. ([25, LEMMAS 1 AND 6]) *For all positive integers ℓ and h , there is an (ℓ, h) -universal tree with at most quasi-polynomial number of leaves. More specifically, the number of leaves is at most $2\ell \binom{\lceil \lg \ell \rceil + h + 1}{h}$, which is polynomial in ℓ if $h = O(\log \ell)$; it is $O(h \ell^{\lg(h/\lg \ell) + 1.45})$ if $h = \omega(\log \ell)$; and—more crudely—it is always $\ell^{\lg h + O(1)}$.*

A natural question then arises, whether one can significantly improve the worst-case upper bounds on the run-time complexity of solving parity games by designing significantly smaller universal trees. We give a negative answer to this question in the next section.

2.4 Smallest universal trees are quasi-polynomial. The main technical result in this section is a quasi-polynomial lower bound on the size of universal ordered trees that matches the upper bound in Theorem 2.2 up to a small polynomial factor. It follows that the smallest universal ordered trees have quasi-polynomial size, and hence the worst-case performance of progress measure lifting algorithms [24, 25] cannot be improved to sub-quasi-polynomial by designing smaller universal ordered trees.¹

THEOREM 2.3. *For all positive integers ℓ and h , every (ℓ, h) -universal tree has at least $\binom{\lceil \lg \ell \rceil + h - 1}{h - 1}$ leaves, which is at least $\ell^{\lg(h/\lg \ell) - 1}$ provided that $2h \leq \ell$.*

This lower bound result shares some similarities with a result of Goldberg and Lifschitz [21], which is for universal trees of a different kind: the height is not bounded and the trees are not ordered.

Proof. First, we give a derivation of the latter bound from the former; we show that

$$\binom{\lceil \lg \ell \rceil + h - 1}{h - 1} = \binom{\lceil \lg \ell \rceil + h - 1}{\lceil \lg \ell \rceil} \geq \ell^{\lg(h/\lg \ell) - 1}$$

provided that $2h \leq \ell$. We start from the inequality $\binom{k}{\ell} \leq \binom{k}{\ell}$ applied to the binomial coefficient

¹The quasi-polynomial lower bound on the size of universal trees has appeared in the technical report [17], which is subsumed by this paper.

$\binom{\lfloor \lg \ell \rfloor + h - 1}{\lfloor \lg \ell \rfloor}$, and take the \lg of both sides. This yields

$$\begin{aligned} \lg \binom{\lfloor \lg \ell \rfloor + h - 1}{\lfloor \lg \ell \rfloor} &\geq \\ &\geq \lfloor \lg \ell \rfloor \cdot \left[\lg (\lfloor \lg \ell \rfloor + h - 1) - \lg \lfloor \lg \ell \rfloor \right] \\ &\geq (\lg \ell - 1) \cdot (\lg h - \lg \lg \ell) \\ &\geq \lg \ell \cdot (\lg(h/\lg \ell) - 1), \end{aligned}$$

where the second inequality follows since $\ell \geq 2$, and the third by the assumption that $2h \leq \ell$.

To prove the first bound, we proceed by induction and show that any (ℓ, h) -universal tree has at least $g(\ell, h)$ leaves, where

$$g(\ell, h) = \sum_{\delta=1}^{\ell} g(\lfloor \ell/\delta \rfloor, h-1),$$

$g(\ell, 1) = \ell$, and $g(1, h) = 1$.

The bounds are clear for $h = 1$ or $\ell = 1$.

Let T be a (ℓ, h) -universal tree, and $\delta \in \{1, \dots, \ell\}$. We claim that the number of nodes at depth $h-1$ of degree greater than or equal to δ is at least $g(\lfloor \ell/\delta \rfloor, h-1)$.

Let T_δ be the subtree of T obtained by removing all leaves and all nodes at depth $h-1$ of degree less than δ : the leaves of the tree T_δ have depth exactly $h-1$.

We argue that T_δ is $(\lfloor \ell/\delta \rfloor, h-1)$ -universal. Indeed, let t be a tree with $\lfloor \ell/\delta \rfloor$ leaves all at depth $h-1$. To each leaf of t we append δ children, yielding the tree t_+ which has $\lfloor \ell/\delta \rfloor \cdot \delta \leq \ell$ leaves all at depth h . Since T is (ℓ, h) -universal, the tree t_+ embeds into T . Observe that the embedding induces an embedding of t into T_δ , since the leaves of t have degree δ in t_+ , hence are also in T_δ .

Let ℓ_δ be the number of nodes at depth $h-1$ with degree exactly δ . So far we proved that the number of nodes at depth $h-1$ of degree greater than or equal to δ is at least $g(\lfloor \ell/\delta \rfloor, h-1)$, so

$$\sum_{i=\delta}^{\ell} \ell_i \geq g(\lfloor \ell/\delta \rfloor, h-1).$$

Thus the number of leaves of T is

$$\sum_{i=1}^{\ell} \ell_i \cdot i = \sum_{\delta=1}^{\ell} \sum_{i=\delta}^{\ell} \ell_i \geq \sum_{\delta=1}^{\ell} g(\lfloor \ell/\delta \rfloor, h-1) = g(\ell, h).$$

It remains to prove that

$$(2.1) \quad g(\ell, h) \geq \binom{\lfloor \lg \ell \rfloor + h - 1}{\lfloor \lg \ell \rfloor}.$$

Define $G(p, h) = g(2^p, h)$ for $p \geq 0$ and $h \geq 1$. Then we have

$$\begin{aligned} G(p, h) &\geq \sum_{k=0}^p G(p-k, h-1), \\ G(p, 1) &\geq 1, \\ G(0, h) &= 1. \end{aligned}$$

To obtain a lower bound on G we define \overline{G} by

$$\begin{aligned} \overline{G}(p, h) &= \overline{G}(p, h-1) + \overline{G}(p-1, h), \\ \overline{G}(p, 1) &= 1, \\ \overline{G}(0, h) &= 1, \end{aligned}$$

so that $G(p, h) \geq \overline{G}(p, h)$. We verify by induction that $\overline{G}(p, h) = \binom{p+h-1}{p}$, which follows from Pascal's identity

$$\binom{p+h-1}{p} = \binom{p+h-2}{p} + \binom{p+h-2}{p-1}.$$

This implies that $G(p, h) \geq \binom{p+h-1}{p}$, from which (2.1) follows.

3 The separation approach

3.1 Languages of play encodings. The outcome of the two players interacting in a parity game by making moves is an infinite path in the game graph. We encode such infinite paths as infinite words over the alphabet $\Sigma_d = \{1, 2, \dots, d\}$ in a natural way: each move—in which an edge e is followed—is encoded by the letter $\pi(e)$, i.e., the priority of edge e .

We write LimsupEven_d for the set of infinite words in which the biggest number that occurs infinitely many times is even, and we write LimsupOdd_d for the set of infinite words in which that number is odd. Observe that sets LimsupEven_d and LimsupOdd_d form a partition of the set $(\Sigma_d)^\omega$ of all infinite words over the alphabet Σ_d . As intended, an infinite play in a parity game graph (of arbitrary size) with edge priorities not exceeding d is winning for Even if and only if the infinite-word encoding of the play is in LimsupEven_d .

Recall that a (parity game) graph is called even if every cycle in it is even (i.e., the highest priority that occurs on the cycle is even). For all positive integers n and d , we define the language $\text{EvenCycles}_{n,d} \subseteq (\Sigma_d)^\omega$ to consist of infinite words that encode an infinite path in an even graph with at most n vertices and d priorities. The languages $\text{EvenCycles}_{n,d}$ can be thought of as finitary under-approximations of the language LimsupEven_d because

$$\text{EvenCycles}_{1,d} \subsetneq \text{EvenCycles}_{2,d} \subsetneq \dots \subsetneq \text{LimsupEven}_d.$$

Languages $\text{OddCycles}_{n,d}$ —that can be thought of as finitary under-approximations of the language LimsupOdd_d —are defined in an analogous way.

3.2 Safety automata and games. The fundamental and simple model that the statement of our main technical result formally refers to is a (non-deterministic) *safety automaton*. Superficially, it closely resembles the classic model of *finite automata*: each safety automaton has a finite set of *states*, a designated *initial state*, and a *transition relation*. (Without loss of generality, we assume that the transition relation is *total*, i.e., for every state s and letter a , there is at least one state s' , such that the triple (s, a, s') is in the transition relation.) The differences between our model of safety automata and the classic model of finite automata with designated *accepting states* are as follows:

- safety automata are meant to accept or reject *infinite words*, not finite ones;
- a safety automaton does not have a designated set of accepting states; instead it has a designated set of *rejecting states*;
- a safety automaton *accepts* an infinite word if there is an infinite run of the automaton on the word in which no rejecting state ever occurs; otherwise it *rejects* the infinite word.

We say that a safety automaton is *deterministic* if the transition relation is a function: for every state s and letter a , there is a unique state s' , such that the triple (s, a, s') is a transition.

Finally, we define the elementary concept of *safety games*, which are played by two players on finite directed graphs in a similar way to parity games, but the goals of the players are simpler than in parity games: the *safety player* wins if a designated set of *unsafe vertices* is never visited, and otherwise the opponent (sometimes called the *reachability player*) wins.

3.3 Safety separating automata. For positive integers n and d , we say that an automaton \mathcal{A} is an (n, d) -*separator* if it is a separator of $\text{EvenCycles}_{n,d}$ and $\text{OddCycles}_{n,d}$; and we say that it is a *strong* (n, d) -*separator* if it is a separator of $\text{EvenCycles}_{n,d}$ and LimsupOdd_d . Note that if an automaton is a strong (n, d) -separator then it is also an (n, d) -separator, but not every (n, d) -separator is strong.

To illustrate the concept of a (strong) (n, d) -separator, we present a simple “multi-counter” strong (n, d) -separator that is implicit in the work of Bernet et al. [1]. We define the automaton $\mathcal{C}_{n,d}$ that, for every odd priority p , $1 \leq p \leq d-1$, keeps a counter c_p that stores the number of occurrences of priority p since the last occurrence of a priority larger than p (even or odd). It is a safety automaton: it rejects a word immediately once the integer stored in any of the $d/2$ counters exceeds n .

In fact, instead of “counting up” (from 0 to n), we prefer to “count down” (from n to 0), which is

equivalent, but it aligns better with the definition of progress measures. More formally, we define the deterministic safety automaton $\mathcal{C}_{n,d}$ in the following way:

- the set of states of $\mathcal{C}_{n,d}$ is the set of $d/2$ -sequences $\langle c_{d-1}, c_{d-3}, \dots, c_1 \rangle$, such that c_p is an integer such that $0 \leq c_p \leq n$ for every odd p , $1 \leq p \leq d-1$; and it also contains an additional *rejecting state* **reject**;
- the initial state is $\langle n, n, \dots, n \rangle$;
- the transition function δ is defined so that $\delta(\text{reject}, p) = \text{reject}$ for all $p \in \Sigma_d$, and $\delta(\langle c_{d-1}, c_{d-3}, \dots, c_1 \rangle, p)$ is equal to:
 - $\langle c_{d-1}, c_{d-3}, \dots, c_{p+1}, n, \dots, n \rangle$ if p is even,
 - $\langle c_{d-1}, c_{d-3}, \dots, c_p - 1, n, \dots, n \rangle$ if p is odd and $c_p > 0$,
 - **reject** if p is odd and $c_p = 0$.

Note that the size of automaton $\mathcal{C}_{n,d}$ is $\Theta(n^{d/2})$.

PROPOSITION 3.1. ([1, 3]) *The safety automaton $\mathcal{C}_{n,d}$ is a strong (n, d) -separator.*

Proof. Firstly, we argue that if the unique run of $\mathcal{C}_{n,d}$ on an infinite word contains an occurrence of the rejecting state then the word is not in $\text{EvenCycles}_{n,d}$. Indeed, the only reason for the unique run of $\mathcal{C}_{n,d}$ to reach the rejecting state is that the state reached after reading some prefix of the word is $\langle c_{d-1}, c_{d-3}, \dots, c_1 \rangle$ with $c_p = 0$ for an odd p , and p is subsequently read. For this to happen, there must be a suffix of the prefix in which there are n occurrences of priority p and no priority higher than p occurs, and the currently read letter is the $(n+1)$ -st occurrence of priority p in the prefix. We argue that if the input word is an encoding of an infinite play in an even graph with at most n vertices then—by the pigeonhole principle—there is a cycle in the graph in which the highest priority is p , which contradicts the assumption that the graph was even. It follows that the infinite word is not in $\text{EvenCycles}_{n,d}$.

Secondly, we argue that if a word is in LimsupOdd_d then the unique run of $\mathcal{C}_{n,d}$ on the word contains an occurrence of the rejecting state. Consider an infinite suffix of the word in which all priorities occur infinitely many times. Unless the unique run reached the rejecting state on the corresponding prefix already, let $\langle c_{d-1}, c_{d-3}, \dots, c_1 \rangle$ be the state reached in the unique run at the end of the prefix. By the assumption that the word is in LimsupOdd_d , the highest priority that occurs in the suffix is odd and it occurs infinitely many times. Take the shortest prefix of the suffix in which the highest priority p occurs $n - c_p + 1$ times. The unique run of $\mathcal{C}_{n,d}$ on the original infinite word reaches the rejecting state upon reading that prefix.

3.4 The separation approach. We now explain how safety separating automata allow to reduce the complex task of solving a parity game to the (conceptually and algorithmically) straightforward task of solving a safety game, by exploiting positional determinacy of parity games. This is the essence of the *separation approach* that implicitly underpins the algorithms of Berret, Janin, and Walukiewicz [1] and of Calude et al. [5], as formalized by Bojańczyk and Czerwiński [3, Chapter 3]. Here, we only consider the simple case of *deterministic* automata. We postpone the discussion of using non-deterministic automata in the separation approach to Sections 4.3 and 4.4, which is the only place where non-determinism seems to be needed.

Given a parity game G with at most n vertices and priorities up to d , and a deterministic safety automaton \mathcal{A} with input alphabet Σ_d , we define a safety game as the *chained product* $G \triangleright \mathcal{A}$, in which

- the dynamics of play and ownership of vertices is inherited from the parity game G ;
- the automaton \mathcal{A} is fed the priorities of the edges corresponding to the moves made by the players;
- the safety winning condition is the safety acceptance condition of the automaton \mathcal{A} .

PROPOSITION 3.2. *If G is a parity game with n vertices and priorities up to d , and if \mathcal{A} is a deterministic safety (n, d) -separator, then Even has a winning strategy in G if and only if she has a winning strategy in the chained-product safety game $G \triangleright \mathcal{A}$.*

Proof. If Even has a winning strategy in the parity game G then—by positional determinacy [12, 28]—she also has one that is positional. We argue that if Even uses such a positional winning strategy in G when playing the chained-product game $G \triangleright \mathcal{A}$, then she also wins the latter. Indeed, the strategy subgraph of the positional winning strategy for Even in G is an even graph with at most n vertices, and hence all words that are fed to the automaton \mathcal{A} are in $\text{EvenCycles}_{n,d}$, and hence they are accepted by \mathcal{A} .

Otherwise, Odd has a positional winning strategy in G , and it can be transferred to a winning strategy for him in $G \triangleright \mathcal{A}$ in the same way as we argued for Even above.

In the rest of the paper, we focus on *strong* (n, d) -separators for two reasons. Firstly—as described in Section 4—all the known quasi-polynomial algorithms for parity games are underpinned by strong (n, d) -separators. Secondly, our proof of the quasi-polynomial lower bound in Section 5 only applies to strong (n, d) -separators.

4 Separating automata everywhere

In this section we argue that the three distinct techniques that have been developed so far for designing quasi-polynomial algorithms for solving parity games can be unified as instances of the separation approach introduced in the previous section. The main unifying aspect that we highlight in this section is that all the three approaches yield constructions of separating automata of quasi-polynomial size, which provides evidence of significance of our main technical result: the quasi-polynomial lower bound on the size of separators (Theorem 5.1) forms a barrier that all of those approaches need to overcome in order to further significantly improve the known complexity of solving parity games. We note that, in contrast to the results of Calude et al. [5] and Lehtinen [27], not all of the proposed quasi-polynomial algorithms explicitly construct separating automata or other objects of (at least) quasi-polynomial size [25, 16], but in the worst case, they too enumerate structures that form the states of the related separating automata (leaves in a universal tree and play summaries, respectively).

In Section 4.1 we generalize the simple separator described in Section 3.3 to a construction that creates strong separators from arbitrary universal trees. This result implies that the main combinatorial structure underlying all the progress measure lifting algorithms [24, 1, 30, 6], including the quasi-polynomial succinct progress measure lifting of Jurdziński and Lazić [25, 11], is intimately linked to the separation approach. In Section 4.2, we briefly discuss the observation of Bojańczyk and Czerwiński [3] that Calude et al.’s [5] play summaries construction can be straightforwardly interpreted as defining a separating automaton; we refer the reader to their very readable technical exposition. Finally, in Sections 4.3 and 4.4, we discuss how to adapt the separation approach to also encapsulate the most recent quasi-polynomial algorithm for solving parity games by Lehtinen [27], based on register games. This requires care because, unlike the constructions based on play summaries and universal trees, separating automata that underpin Lehtinen’s reduction from parity games to register games seem to require non-determinism, and in general, Proposition 3.2 does not hold for non-deterministic automata.

4.1 Separating automata from universal trees.

For positive integers n and d , such that d is even, let $L_{n,d/2}$ be the set of leaves in an $(n, d/2)$ -universal tree. The definition of the deterministic safety automaton $\mathcal{U}_{n,d}$ bears similarity to the definition of the simple “multi-counter” separator $\mathcal{S}_{n,d}$ from Section 3.3. Again, the states are $d/2$ -sequences of “counters”, but

the “counting down” is done in a more abstract way than in $\mathcal{S}_{n,d}$, using instead the natural lexicographic order on the nodes of the universal tree.

More formally, we define a deterministic safety automaton $\mathcal{U}_{n,d}$ in the following way:

- the set of states of $\mathcal{U}_{n,d}$ is the set $L_{n,d/2}$ of leaves in the $(n, d/2)$ -universal tree together with an additional **reject** state;
- the initial state is the largest leaf (in the lexicographic tree order) and the only rejecting state is **reject**;
- the transition function δ is defined so that $\delta(\mathbf{reject}, p) = \mathbf{reject}$ for all $p \in \Sigma_d$, and if $s \neq \mathbf{reject}$ then $\delta(s, p)$ is equal to:
 - the largest leaf s' such that $s|_p = s'|_p$ if p is even,
 - the largest leaf s' such that $s|_p > s'|_p$ if p is odd,
 - **reject** if p is odd and no such leaf exists.

REMARK 4.1. *Let $\mathcal{T}_{n,d}$ be an ordered tree in which all the leaves have depth $d/2$ and every non-leaf node has exactly n children; note that $\mathcal{T}_{n,d}$ is trivially an $(n, d/2)$ -universal tree. An instructive exercise that we recommend is to compare the structures of the automaton $\mathcal{U}_{n,d}$ based on the $(n, d/2)$ -universal tree $\mathcal{T}_{n,d}$ and of the simple separating automaton $\mathcal{S}_{n,d}$ from Section 3.3.*

THEOREM 4.1. *For every $(n, d/2)$ -universal tree, the safety automaton $\mathcal{U}_{n,d}$ is a strong (n, d) -separator.*

Proof. First we prove that if an infinite word $w \in \Sigma_d$ is in $\text{EvenCycles}_{n,d}$ then it is accepted by the safety automaton $\mathcal{U}_{n,d}$. Let G be an even graph with at most n vertices and priorities up to d , in which w occurs as an encoding of an infinite path; let $\langle v_1, v_2, v_3, \dots \rangle$ be the sequence of vertices in the infinite path in G . By Theorem 2.1 and Proposition 2.1, there is a progress measure μ that maps vertices in G into the set of leaves $L_{n,d/2}$ in the $(n, d/2)$ -universal tree. Consider the unique run $\langle q_1, q_2, q_3, \dots \rangle$ of $\mathcal{U}_{n,d}$ on the word w , where q_1 is the initial state of $\mathcal{U}_{n,d}$ and we have $\delta(q_i, \pi(v_i, v_{i+1})) = q_{i+1}$ for all positive integers i . By the definition of the initial state in $\mathcal{U}_{n,d}$, we have that $q_1 \geq \mu(v_1)$, and the definition of the transition function of $\mathcal{U}_{n,d}$ allows to establish—by a straightforward induction on i —that $q_i \geq \mu(v_i)$ for all positive integers i . It follows that the run never reaches the **reject** state, and hence the run is accepting.

Now we argue that for every infinite word w in LimsupOdd_d , the unique run of $\mathcal{U}_{n,d}$ on it eventually reaches the **reject** state. Our argument is a generalization of the corresponding one in the proof of Proposition 3.1. Consider an infinite suffix $x = \langle x_1, x_2, x_3, \dots \rangle$

of the word w in which all priorities occur infinitely many times. Unless the unique run reached the rejecting state on the corresponding prefix already, let s_1 be the state reached in the unique run at the end of the prefix. Let p be the highest priority that occurs in x ; by the assumption that w is in LimsupOdd_d , the priority p is odd. Let $\langle s_1, s_2, s_3, \dots \rangle$ be the unique run of $\mathcal{U}_{n,d}$ on x : for every positive integer i , we have $\delta(s_i, x_i) = s_{i+1}$. Note that $\langle s_1, s_2, s_3, \dots \rangle$ is a suffix of the unique run of $\mathcal{U}_{n,d}$ on w . By the definition of the transition function of $\mathcal{U}_{n,d}$, if s_i and s_{i+1} are not equal to the **reject** state then $s_i|_p \geq s_{i+1}|_p$, and if $x_i = p$ then $s_i|_p > s_{i+1}|_p$. Therefore, unless $s_i = \mathbf{reject}$ for some positive integer i , there is an infinite subsequence of $\langle s_1, s_2, s_3, \dots \rangle$ whose p -truncations form a strictly decreasing sequence, which contradicts finiteness of the set $L_{n,d/2}$ of states of $\mathcal{U}_{n,d}$.

The following can be obtained by using the nearly optimal quasi-polynomial $(n, d/2)$ -universal trees from Theorem 2.2 in the construction of the automaton $\mathcal{U}_{n,d}$.

COROLLARY 4.1. *There are deterministic safety automata that are strong (n, d) -separators of size $n^{\lg d + O(1)}$.*

4.2 Separating automata from play summaries.

Bojańczyk and Czerwiński [3, Chapter 3] give an accessible exposition of how the breakthrough result of Calude et al. [5] can be viewed as a construction of a deterministic automaton of quasi-polynomial size that separates $\text{EvenLoops}_{n,d}$ from LimsupOdd_d , which implies separation of $\text{EvenCycles}_{n,d}$ from LimsupOdd_d .

One superficial difference between our exposition of separators and theirs is that we use the model of safety automata, while they consider the dual model of *reachability automata* instead. (In reachability automata, an infinite word is accepted if and only if one of the designated *accepting states* is reached; otherwise it is rejected.) If, in Bojańczyk and Czerwiński’s construction, we swap the roles of players Even and Odd, and we make the accepting states rejecting, we get a safety automaton that separates $\text{EvenCycles}_{n,d}$ from LimsupOdd_d .

THEOREM 4.2. ([5, 3]) *The play summaries data structure of Calude et al. yields deterministic safety automata that are strong (n, d) -separators of size $n^{\lg d + O(1)}$.*

4.3 Non-deterministic automata and the separation approach.

The possible usage of non-deterministic automata in the separation approach to solving parity games is less straightforward. First of all, the game dynamics needs to be modified to explicitly include the choices that resolve non-determinism in every

step. We give the power to make those choices to Even, but this extra power does not suffice to make her win the chained-product game whenever she has a winning strategy in the original parity game. The reason for this failure in transferring winning strategies from the parity game to the chained-product safety game is that in arbitrary non-deterministic automata it may be impossible to successfully resolve non-deterministic choices at a position in the input word without knowing the letters at the later positions. In the game, however, the play is a result of the future choices of both the player and her opponent, and the former cannot predict the latter.

A well-known example of non-deterministic automata for which the chained-product game is equivalent to the original game is the class of *good-for-games* automata [22]. They have the desired property that the non-deterministic choices of the automaton can always be resolved based only on the letters in the word at the positions up to the current one, thus making it possible to continue constructing an accepting run for all words accepted by the non-deterministic automaton that have the word read so far as a prefix.

We consider a weaker property than the one mentioned above, one that is sufficient for strategy transfer in the context of the separation approach. We say that a non-deterministic automaton \mathcal{A} is a *good-for-separation* strong (n, d) -separator if for every parity game G with at most n vertices and d priorities, in the chained-product game $G \triangleright \mathcal{A}$, there is a way to resolve the non-deterministic choices of \mathcal{A} based only on the prefix of the play so far, in such a way that a construction of an accepting run can be continued for all words in $\text{EvenCycles}_{n,d}$ (but not necessarily for all the words accepted by \mathcal{A}).

More formally, a non-deterministic automaton \mathcal{A} is a *good-for-separation* strong (n, d) -separator if it rejects every word in LimsupOdd_d , and for every even game G with n vertices and priorities up to d , there is a winning strategy for Even in the chained-product game $G \triangleright \mathcal{A}$. We note that the automata that can be derived using Lehtinen’s techniques [27] are good-for-separation and hence they are suitable for being used in the separation approach.

Note that our quasi-polynomial lower bound in Section 5 holds for all non-deterministic strong (n, d) -separators, regardless of their suitability for strategy transfer and for solving parity games using the separation approach.

4.4 Separating automata from register games.

First, we recall the definition of a (non-deterministic) *parity automaton*. Like safety automata, parity automata process infinite words, but instead of having des-

ignated rejecting states, every transition has a *priority*, which is a positive integer. The set of transitions consists of tuples of the form (s, a, p, s') : such a transition has priority p , and when reading a letter a in state s , the automaton moves to state s' . We say that a run of a parity automaton on an infinite word is *accepting* if the highest transition priority that occurs infinitely often is even; otherwise it is *rejecting*. The automaton *recognizes* the language of all words on which it has an accepting run. The following observation is straightforward.

PROPOSITION 4.1. *For every positive integer d , there is a deterministic parity automaton \mathcal{P}_d , with 1 state and d priorities, that recognizes the language LimsupEven_d .*

Indeed, it suffices to equip the transition relation \mathcal{P}_d with transitions (s, p, p, s) , where s is the unique state of \mathcal{P}_d , for all $p \in \Sigma_d$. Observe that *recognizing* the language LimsupEven_d implies being a strong (n, d) -separator for *all* positive integers n : a much stronger property than being a strong (n, d) -separator for *some* positive integer n . Since the automaton \mathcal{P}_d is deterministic, it is in principle suitable for the separation approach applied to games with d distinct priorities. Note, however, that using it brings no tangible benefit for solving parity games: the chained-product game $G \triangleright \mathcal{P}_d$ has as few vertices as the original game G , but its number of distinct priorities is no smaller.

We argue that the key technical result in the recent work of Lehtinen [27] can be interpreted as proving the following theorem.

THEOREM 4.3. ([27]) *For all positive integers n and d , there is a good-for-separation parity automaton $\mathcal{R}_{n,d}$, with at most $\binom{d+\lceil \lg n \rceil + 1}{d} = n^{\lg d + O(1)}$ states and $2\lceil \lg n \rceil + 3$ priorities, that is a strong (n, d) -separator.*

Note that this theorem does indeed bring tangible benefits for solving parity games (via the separation approach) because if a parity game G has n vertices and d priorities, then the chained-product game $G \triangleright \mathcal{R}_{n,d}$ has only a quasipolynomial number of vertices and a logarithmic number of priorities. Even if an unsophisticated algorithm—with run-time that is exponential in the number of priorities—is used to solve the chained-product game $G \triangleright \mathcal{R}_{n,d}$, the overall run-time is quasipolynomial.

For every parity game G , Lehtinen defines the corresponding *register game* R_G , whose vertices consist of vertices of game G together with $\lfloor 1 + \lg n \rfloor$ -sequences $\langle r_{\lfloor 1 + \lg n \rfloor}, \dots, r_2, r_1 \rangle$ of the so-called *registers* that hold priorities, i.e., numbers from the set $\{1, 2, \dots, d\}$. The game is played on a copy of G in the usual way, additionally at her every move player Even is given a

chance—but not an obligation—to “reset” one of the registers, and each register always holds the biggest priority that has occurred since it was last reset.

What needs explaining is what “resetting a register” entails. When the register at position k is reset, then the next register sequence is $\langle r_{\lfloor 1+\lg n \rfloor}, \dots, r_{k+1}, r_{k-1}, \dots, r_1, 1 \rangle$, that is registers at positions 1 to $k-1$ are promoted to positions 2 to k , respectively, and the just-reset register is now at position 1 and it has value 1. Moreover—and very importantly for Lehtinen’s construction—resetting the register at position k causes the even priority $2k$ to occur in game R_G if the value in the register was even, and the odd priority $2k+1$ otherwise. If, instead, Even decides not to reset any register then the odd priority 1 occurs.

Lehtinen’s main technical result is that the original parity game G and the register game R_G have the same winners. She proves it by arguing that if Even has a (positional) winning strategy in G then she has a strategy of resetting registers in R_G so that she again wins the parity condition (albeit with the number of priorities reduced from an arbitrarily large d in G to only $\lfloor 1+\lg n \rfloor$ in R_G). Our approach is to separate the graph structure from Lehtinen’s mechanism to capture the original parity winning condition using registers. We now define an automata-theoretic analogue of her construction in which we use non-determinism to model the ability to pick various resetting strategies.

For all positive numbers n and d , such that d is even, we define the *non-deterministic parity automaton* $\mathcal{R}_{n,d}$ in the following way.

- The set of states of $\mathcal{R}_{n,d}$ is the set of non-increasing $\lfloor 1+\lg n \rfloor$ -sequences $\langle r_{\lfloor 1+\lg n \rfloor}, \dots, r_2, r_1 \rangle$ of “registers” that hold numbers in $\{1, 2, \dots, d\}$. The initial state is $\langle 1, 1, \dots, 1 \rangle$.
- For every state $s = \langle r_{\lfloor 1+\lg n \rfloor}, \dots, r_2, r_1 \rangle$ and letter $p \in \Sigma_d$, we define the *update of s by p* to be the state $\langle r_{\lfloor 1+\lg n \rfloor}, \dots, r_k, p, \dots, p \rangle$, where k is the smallest such that $r_k > p$.
- For every state $s = \langle r_{\lfloor 1+\lg n \rfloor}, \dots, r_2, r_1 \rangle$ and for every k , $1 \leq k \leq \lfloor 1+\lg n \rfloor$, we define the *k -reset of s* to be the state $\langle r_{\lfloor 1+\lg n \rfloor}, \dots, r_{k+1}, r_{k-1}, \dots, r_2, 1 \rangle$.
- For every state s and letter $p \in \Sigma_d$, there is a transition $(s, p, 1, s')$ of priority 1 in the transition relation, and where s' is the update of s by p ; we call this a *non-reset* transition.
- For every state s , letter $p \in \Sigma_d$, and for every k , $1 \leq k \leq \lfloor 1+\lg n \rfloor$, if $s'' = \langle r_{\lfloor 1+\lg n \rfloor}, \dots, r_2, r_1 \rangle$ is the update of s by p and r_k is even, then there is a transition $(s, p, 2k, s')$ of priority $2k$ in the transition relation, where s' is the k -reset of s'' . We say that this transition is an *even reset of register k* .
- For every state s , letter $p \in \Sigma_d$, and for every k ,

$1 \leq k \leq \lfloor 1+\lg n \rfloor$, if $s'' = \langle r_{\lfloor 1+\lg n \rfloor}, \dots, r_2, r_1 \rangle$ is the update of s by p and r_k is odd, then there is a transition $(s, p, 2k+1, s')$ of priority $2k+1$ in the transition relation, where s' is the k -reset of s'' . We say that this transition is an *odd reset of register k* .

The states of $\mathcal{R}_{n,d}$ are monotone sequences of length $\lfloor 1+\lg n \rfloor$, consisting of positive integers no larger than d , and hence their number is bounded by $\binom{d+\lfloor \lg n \rfloor+1}{d} \leq d^{\lfloor 1+\lg n \rfloor}$, which is $n^{\lg d + O(1)}$. We note that the register game R_G that Lehtinen constructs from a parity game G is essentially the same as the chained-product game $G \triangleright \mathcal{R}_{n,d}$.

The proof that the parity automaton $\mathcal{R}_{n,d}$ is a strong (n, d) -separator can be obtained by adapting Lehtinen’s proofs showing that the *register index* of a parity game with n vertices is at most $1+\lg n$ [27, Theorem 4.7] and that a winning strategy for Odd in a parity game G yields a winning strategy for him in the corresponding register game R_G [27, Lemma 3.3]. We refer an interested reader to Lehtinen’s paper for the details. We note, however, that the strategy for resetting registers in the register game R_G (and hence also in the chained-product game $G \triangleright \mathcal{R}_{n,d}$) that Lehtinen constructs in the proof of [27, Theorem 4.7] depends only on the current content of the registers and the current vertex in the game. This property implies that the automata $\mathcal{R}_{n,d}$ are good-for-separation (n, d) -separators in the sense discussed in Section 4.3.

We stress that the quasi-polynomial lower bound for the size of strong separators that we establish in Section 5 applies to *safety* automata, but not to *parity* automata such as $\mathcal{R}_{n,d}$.

On the other hand, we argue that the parity automata $\mathcal{R}_{n,d}$ can be turned into good-for-separation *safety* (n, d) -separators by taking a chained product with other deterministic safety separators. Let \mathcal{R} be a parity automaton over the alphabet Σ_d and with d' transition priorities, and let \mathcal{S} be a safety automaton over the alphabet $\Sigma_{d'}$. Consider the following chained-product automaton $\mathcal{R} \triangleright \mathcal{S}$.

- The set of states is the set of pairs (r, s) , where r is a state of \mathcal{R} and s is a state of \mathcal{S} . The initial state is the pair of the initial states of the two automata. A state (r, s) is rejecting if s is rejecting in \mathcal{S} .
- If (r, a, p, r') is a transition in \mathcal{R} , where $a \in \Sigma_d$ is the letter read by the transition and $p \in \Sigma_{d'}$ is the priority of the transition, and if (s, p, s') is a transition in \mathcal{S} (for deterministic automata, we often write $\delta(s, p) = s'$ instead), then $((r, s), a, (r', s'))$ is a transition in $\mathcal{R} \triangleright \mathcal{S}$.

Let $n' = \binom{d+\lfloor \lg n \rfloor+1}{d} = n^{\lg d + O(1)}$ be the number of states in the parity register automaton $\mathcal{R}_{n,d}$ and let $d' = 2\lfloor \lg n \rfloor + 4$ be an upper bound on its priorities.

PROPOSITION 4.2. *If \mathcal{S} is a deterministic safety strong (n', d') -separator then the chained-product automaton $\mathcal{R}_{n,d} \triangleright \mathcal{S}$ is a safety good-for-separation strong (n, d) -separator.*

Proof. Let G be an even graph with n vertices and priorities up to d . Careful inspection of Lehtinen’s proof of [27, Theorem 4.7] reveals that Even has a *positional* winning strategy σ in the chained-product game $G \triangleright \mathcal{R}_{n,d}$. Note that the strategy subgraph G' of that strategy is even, and that it has at most n' vertices and priorities up to d' . Since \mathcal{S} is a deterministic (n', d') -separator, on every path in G' the unique run of \mathcal{S} on the sequence of priorities that occur on the path is accepting. It follows that in the game $(G \triangleright \mathcal{R}_{n,d}) \triangleright \mathcal{S}$, using strategy σ on the $G \triangleright \mathcal{R}_{n,d}$ component guarantees that Even always wins, and hence Even has a winning strategy in the equivalent game $G \triangleright (\mathcal{R}_{n,d} \triangleright \mathcal{S})$.

On the other hand, every word in LimsupOdd_d is rejected by $\mathcal{R}_{n,d}$ because it is a strong (n, d) -separator. It follows that in every run of $\mathcal{R}_{n,d}$ on such a word, the highest transition priority that occurs infinitely often is odd. In other words, in every run of the chained-product automaton $\mathcal{R}_{n,d} \triangleright \mathcal{S}$ on such a word, the word over the alphabet $\Sigma_{d'}$ that is fed into the automaton \mathcal{S} is in $\text{LimsupOdd}_{d'}$, and hence it is rejected because \mathcal{S} is a strong (n', d') -separator.

Consider taking \mathcal{S} to be the deterministic safety automaton $\mathcal{U}_{n',d'}$ from Theorem 4.1. By Proposition 4.2, the chained-product automaton $\mathcal{R}_{n,d} \triangleright \mathcal{U}_{n',d'}$ is a safety good-for-separation (n, d) -separator. Moreover, if $\mathcal{U}_{n',d'}$ is based on the nearly optimal quasi-polynomial universal trees from Theorem 2.2 then the size of $\mathcal{R}_{n,d} \triangleright \mathcal{U}_{n',d'}$ is $n^{\lg n(\lg d + O(1))}$.

5 Universal trees inside separating automata

The main result established in this section, and a main technical contribution of the paper, is the following quasi-polynomial lower bound on the size of all safety strong separators.

THEOREM 5.1. *Every non-deterministic safety strong (n, d) -separator has at least $\binom{\lfloor \lg n \rfloor + d/2 - 1}{\lfloor \lg n \rfloor}$ states, which is at least $n^{\lg(d/\lg n) - 2}$.*

Since the currently known quasipolynomial time algorithms for solving parity games implicitly or explicitly construct a separating automaton [5, 25, 16, 27], the size of which dictates the complexity, this result explains in a unified way the quasipolynomial barrier. Moreover, our proof of the theorem pinpoints universal trees as the underlying combinatorial structure behind all the recent

algorithms, by establishing that there is an $(n, d/2)$ -universal tree hiding in the set of states of every safety strong (n, d) -separator.

LEMMA 5.1. *If \mathcal{A} is a (non-deterministic) safety strong (n, d) -separator then there is an injective mapping from the leaves of some $(n, d/2)$ -universal tree into the states of \mathcal{A} .*

Note that Theorem 5.1 follows from Lemma 5.1 by applying Theorem 2.3—our quasi-polynomial lower bound for universal trees, because $d \leq n$.

We prove Lemma 5.1 in two steps. In the first step, we show that every safety strong (n, d) -separator has a *tree-like* structure (Lemma 5.2). Then, assuming this special structure, we prove that there is a universal tree whose leaves can be injectively mapped into the set of states of \mathcal{A} (Lemma 5.3).

5.1 Tree-like structure. A binary relation \preceq on a set X is called a *linear quasi-order* if it is reflexive, transitive, and total (i.e. such that for all $x, y \in X$ either $x \preceq y$, or $y \preceq x$, or both). If $x \preceq y$ and $y \not\preceq x$, then we write $x \prec y$. An equivalence class of \preceq is a maximal set $e \subseteq X$ such that $x \preceq y$ and $y \preceq x$ for all $x, y \in e$. It is well-known that the equivalence classes of \preceq form a partition of X and given two equivalence classes e and e' , there exist $x \in e$ and $y \in e'$ such that $x \preceq y$ if and only if for all $x \in e$ and $y \in e'$, $x \preceq y$. When this is the case, it is denoted by $e \preceq e'$, and $e \prec e'$ when additionally $e' \not\preceq e$.

Given two linear quasi-orders \preceq_1 and \preceq_2 , we write $\preceq_1 \subseteq \preceq_2$ if for all $x, y \in X$, $x \preceq_1 y$ implies $x \preceq_2 y$. In that case, any equivalence class of \preceq_2 is formed with a partition of equivalence classes of \preceq_1 . In other words, an equivalence class of \preceq_1 is included in a unique equivalence class of \preceq_2 and disjoint from the other ones.

For an automaton \mathcal{A} over the alphabet Σ_d , a *tree-decomposition* of \mathcal{A} is a sequence of linear quasi-orders $\preceq_1 \subseteq \preceq_3 \subseteq \dots \subseteq \preceq_{d+1}$ on the set of non-rejecting states of \mathcal{A} such that:

1. if (s, p, s') is a transition in \mathcal{A} then $s \succeq_{2i+1} s'$ for all i such that $p < 2i + 1 \leq d + 1$.
2. if (s, p, s') is a transition in \mathcal{A} and p is odd then $s \succ_{2i+1} s'$ for all i such that $1 \leq 2i + 1 \leq p$.
3. \preceq_{d+1} has a single equivalence class, containing all non-rejecting states of \mathcal{A} .

In other words, reading a priority cannot cause an increase with respect to orders with indices greater than it, and additionally reading an odd priority necessarily causes a decrease with respect to orders whose indices are smaller than or equal to this priority. If there is a tree-decomposition of \mathcal{A} , we say that \mathcal{A} is *tree-like*. Given a tree decomposition D of \mathcal{A} , we define the *D-tree*

of \mathcal{A} , denoted $\text{tree}_D(\mathcal{A})$, as follows (recall the notation for ordered trees from Section 2.2):

- nodes of the $\text{tree}_D(\mathcal{A})$ are sequences $\langle e_{d-1}, e_{d-3}, \dots, e_p \rangle$, where p is odd, $1 \leq p \leq d-1$, and where every branching direction e_i is an equivalence class of the quasi-order \preceq_i , such that $e_{d-1} \supseteq e_{d-3} \supseteq \dots \supseteq e_p$,
- the order between branching directions e_i, e'_i being equivalence classes of \preceq_i is $e_i < e'_i$ when $e_i \prec_i e'_i$.

Notice that for a non-rejecting state q of \mathcal{A} , there is a unique sequence $e_{d-1} \supseteq e_{d-3} \supseteq \dots \supseteq e_1$ where for every i , e_i is an equivalence class of \preceq_i containing q . One can thus assign a non-rejecting state q to the corresponding leaf $\langle e_{d-1}, e_{d-3}, \dots, e_1 \rangle$ in such a way that for every odd priority $p \in \{1, 3, \dots, d-1\}$, $q \prec_p q'$ if and only if the p -truncation of the leaf assigned to q is smaller in the lexicographic order than the p -truncation of the leaf assigned to q' .

An automaton is *accessible* if for every state q there exists a run from an initial state to q , and moreover, if q is non-rejecting, there exists such a run which does not go through any rejecting state. The first of the two steps of the proof of Lemma 5.1 can be summarized by the following lemma.

LEMMA 5.2. *Every accessible non-deterministic safety strong (n, d) -separator is tree-like.*

Proof. We define the linear quasi-orders inductively starting from higher indices, that is, in the order $\preceq_{d+1}, \preceq_{d-1}, \dots, \preceq_1$. As \preceq_{d+1} we take the quasi-order in which all non-rejecting states are in a single equivalence class. Condition 3. is already satisfied.

Assume now that the quasi-orders $\preceq_{d+1}, \preceq_{d-1}, \dots, \preceq_{2i+3}$ are already defined, and that they fulfil Conditions 1. and 2. We define \preceq_{2i+1} as a refinement of \preceq_{2i+3} . If $q \prec_{2i+3} r$ then we set $q \prec_{2i+1} r$ as well. So it remains to define whether \preceq_{2i+1} holds for states q and r in the same equivalence class of \preceq_{2i+3} . Before this, notice that we are already guaranteed that \preceq_{2i+1} satisfies Conditions 1. and 2. for priorities higher than $2i+1$. Indeed, Condition 1. talks only about priorities smaller than $2i+1$. By Condition 2. applied to \preceq_{2i+3} we know that if priority p is odd, $p \geq 2i+3$ (i.e., $p > 2i+1$), and (s, p, s') is a transition, then $s \succ_{2i+3} s'$, so also $s \succ_{2i+1} s'$. Therefore we only need to define \preceq_{2i+1} in such a way that Condition 1. is satisfied for priorities from the set $\{1, \dots, 2i\}$ and Condition 2. is satisfied for the priority $2i+1$. Intuitively speaking, now we only have to care about priorities that are at most $2i+1$.

Let e be an arbitrary equivalence class of \preceq_{2i+3} . Consider an automaton $\mathcal{A}_{e, 2i+1}$, which contains a part of \mathcal{A} consisting of only the states that belong to the class e , and only the transitions that have endpoints inside e and are labelled by priorities at most $2i+1$ (notice that $\mathcal{A}_{e, 2i+1}$ may not be complete). Observe now that there cannot be any odd cycle in $\mathcal{A}_{e, 2i+1}$. Indeed, otherwise we could consider the infinite run that first reaches this cycle from an initial state in \mathcal{A} without visiting any rejecting state (which is possible under the assumption that \mathcal{A} is accessible), and then goes around this cycle forever (note that none of the states in the cycle is rejecting); the word read by this run would be in LimsupOdd_d , but it would be accepted by the automaton (and it should not). Notice that this is exactly the point in the proof of Lemma 5.2 where we need the assumption about rejecting all the words from LimsupOdd_d . Rejecting all the words from $\text{OddCycles}_{n,d}$ would not be sufficient, as the word described above may not arise from an odd graph of size n , and thus does not have to belong to $\text{OddCycles}_{n,d}$. Therefore, on every path in $\mathcal{A}_{e, 2i+1}$, at most $|\mathcal{A}_{e, 2i+1}| - 1$ edges are labelled by letters with priority $2i+1$. Let the *resistance* of a state in $\mathcal{A}_{e, 2i+1}$ be the maximal number of edges labelled by priority $2i+1$ over all paths starting in that state. By the previous observation, the resistance of a state is always finite. Having defined the resistance, for two states s and s' in $\mathcal{A}_{e, 2i+1}$, we say that $s \preceq_{2i+1} s'$ if the resistance of s is not greater than the resistance of s' .

We have to show that such a definition of \preceq_{2i+1} indeed fulfils Conditions 1. and 2. For Condition 1. we have to check that letters with priority smaller than $2i+1$ never cause an increase in the quasi-order \preceq_{2i+1} . Consider priority p , such that $p < 2i+1$ and (s, p, s') is a transition. We know by the induction assumption that $s \succeq_{2i+3} s'$. If $s \succ_{2i+3} s'$ then as well $s \succ_{2i+1} s'$, and we are done with Condition 1. Otherwise, s and s' are in the same equivalence class of \preceq_{2i+3} , and it is enough to show that s has resistance not greater than that of s' . However, if s' has resistance k and (s, p, s') is a transition, then s also has resistance at least k . Thus, state s cannot have smaller resistance than s' . This implies that indeed $s \succeq_{2i+1} s'$, and Condition 1. is fulfilled. For Condition 2., we need to show that reading priority $2i+1$ causes a decrease with respect to \preceq_{2i+1} . Assume that there is a transition $(s, 2i+1, s')$. By the inductive hypothesis for \prec_{2i+3} , Condition 1. implies that $s \succeq_{2i+3} s'$. If $s \succ_{2i+3} s'$ then we are done as before, so assume that s and s' are in the same equivalence class of \preceq_{2i+3} . In order to show that $s \succ_{2i+1} s'$ it is enough to observe that s has greater resistance than s' , which establishes Condition 2.

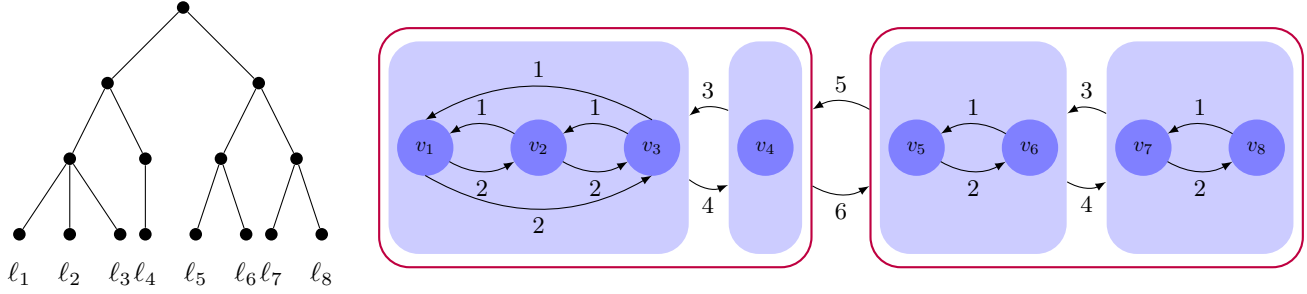


Figure 1: An ordered tree t and the corresponding even graph G_t .

5.2 Universal tree. The main goal of this section is to complete the proof of Lemma 5.1 by showing that the tree-like structure of every safety strong (n, d) -separator uncovered in the previous section has hidden inside it an $(n, d/2)$ -universal tree, whose leaves can be injectively mapped into states of the separator.

Let \mathcal{A} be a safety strong (n, d) -separator. First note that we can assume that \mathcal{A} is accessible, by removing the states that are not reachable from initial states and by making rejecting the non-rejecting states which are only reachable by visiting a rejecting state. By Lemma 5.2 we know that there is a tree-decomposition D of \mathcal{A} . We are going to prove that $\text{tree}_D(\mathcal{A})$ is a $(n, d/2)$ -universal tree and that there is an injective mapping from its leaves into the states of \mathcal{A} .

By definition, there is a bijection between the leaves of $\text{tree}_D(\mathcal{A})$ and the equivalence classes of \preceq_1 , mapping a leaf $\langle e_{d-1}, e_{d-3}, \dots, e_1 \rangle$ to e_1 . For every equivalence class e of \preceq_1 , pick a state q_e in e . Consider the function mapping a leaf $\langle e_{d-1}, e_{d-3}, \dots, e_1 \rangle$ of $\text{tree}_D(\mathcal{A})$ to the state q_{e_1} . This function is injective from the leaves of $\text{tree}_D(\mathcal{A})$ into the states of the automaton. Thus, in order to complete the proof of Lemma 5.1, it suffices to prove the following lemma.

LEMMA 5.3. *If \mathcal{A} is a safety strong (n, d) -separator then for every tree-decomposition D of \mathcal{A} , the D -tree of \mathcal{A} is $(n, d/2)$ -universal.*

Proof. It is enough to show that every tree t of height at most $d/2$ and with at most n leaves can be isomorphically embedded into $\text{tree}_D(\mathcal{A})$. Without loss of generality, we assume that t has exactly n leaves, and that all the leaves have depth $d/2$. Otherwise, if the number of leaves is less than n , add some branches to the tree so as to have exactly n leaves; and if a leaf is not at depth $d/2$, add a path from the leaf so as to reach depth $d/2$. If such a tree can be isomorphically embedded in $\text{tree}_D(\mathcal{A})$ then so can the original one.

As a running example, consider the ordered tree t with $n = 8$ and $d = 6$ in Figure 1.

The proof proceeds by the following three steps.

1. The construction of an even graph G_t with n vertices and d priorities, whose structure reflects the structure of the ordered tree t and, in particular, whose vertices correspond to leaves in t .
2. The construction of an infinite word α_t that is the priority projection of an infinite path in G_t (and that hence is in $\text{EvenCycles}_{n,d}$), whose sufficiently long finite prefix is highly repetitive.
3. A proof—using the highly repetitive nature of word α_t —that t can be isomorphically embedded into $\text{tree}_D(\mathcal{A})$.

Construction of G_t . As the set of vertices of G_t we take $\{v_1, v_2, \dots, v_n\}$. There is a vertex v_i for each leaf l_i in tree t , for every $i = 1, 2, \dots, n$, where l_1, l_2, \dots, l_n are all the leaves of tree t listed in the (increasing) lexicographic order. Consider i and j , such that $1 \leq i < j \leq n$, and let p be the smallest priority in $\{1, 2, \dots, d\}$, such that $l_i|_p = l_j|_p$. Note that then $l_i|_p = l_j|_p$ is the maximum-depth common ancestor of leaves l_i and l_j , and p is even. In graph G_t , there is then an edge from v_i to v_j with the even priority p , and there is an edge from v_j to v_i with the odd priority $p - 1$. Moreover, for every vertex v_i and every even priority p , $1 < p \leq d$, there is a self-loop from v_i to itself in G_t with the even priority p . Note that every cycle in graph G_t is even.

The right-hand-side of Figure 1 illustrates the graph G_t for our running example tree t illustrated in the left-hand-side of the figure. In order to avoid clutter, a single edge drawn in Figure 1 from one set of vertices (enclosed in a rounded rectangle) to another denotes the set of edges from every vertex in the former set to every vertex in the latter; moreover, the self-loops are not shown.

Construction of α_t . We now describe an infinite path in G_t whose priority projection gives an infinite word α_t ; note that the latter is in $\text{EvenCycles}_{n,d}$ because G_t

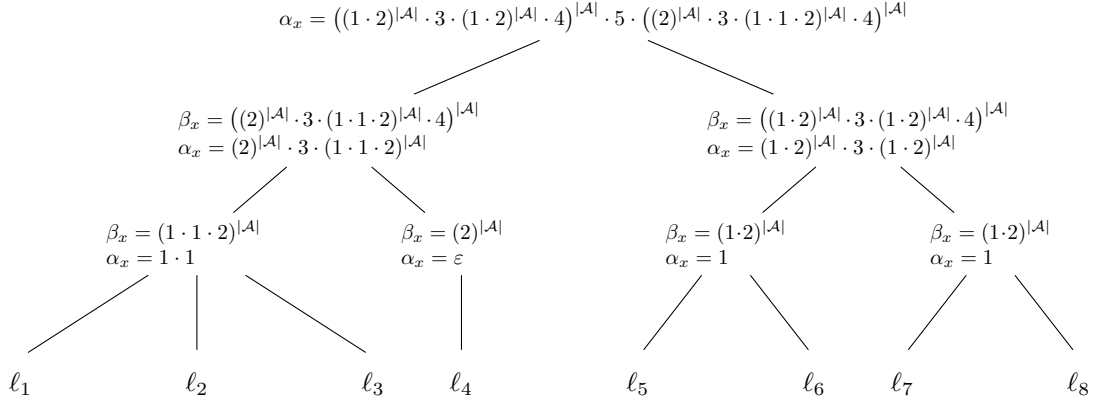


Figure 2: Words α_x and β_x for every non-leaf node x in the ordered tree t from Figure 1.

is an even graph. Before we give the rigorous and general construction, we illustrate the construction on our running example. In the case of graph G_t in Figure 1, the infinite path is as follows.

1. Follow the cycle between vertices v_8 and v_7 a large number of times, alternating between edge priorities 1 and 2.
2. Reach vertex v_6 from vertex v_7 with edge priority 3.
3. Follow the cycle between vertices v_6 and v_5 a large number of times, alternating between edge priorities 1 and 2.
4. Reach vertex v_8 from vertex v_5 with edge priority 4.
5. Repeat steps 1, 2, 3 and 4 a large number of times.
6. Reach vertex v_4 from vertex v_5 with edge priority 5.
7. Take the self-loop around vertex v_4 , with edge priority 2, a large number of times.
8. Reach vertex v_3 from vertex v_4 with edge priority 3.
9. Follow the cycle going through vertices v_3 , v_2 , and v_1 a large number of times, alternating between edge priorities 1, 1, and 2.
10. Reach vertex v_4 from vertex v_1 with edge priority 4.
11. Repeat steps 7., 8., 9., and 10. a large number of times.
12. After all those steps, in order to make the path and the word infinite, once vertex v_1 is reached after step 9., take the self-loop around v_1 , with edge priority 2, infinitely many times.

Now we give a general and rigorous definition of an infinite word α_t for an arbitrary ordered tree t and the corresponding even graph G_t . For every node x of t , we define two finite words α_x and β_x inductively. If x is a leaf then we set $\alpha_x = \beta_x = \varepsilon$. Let x be a node of depth $d/2 - k$, $0 < k \leq d/2$, and let x_1, x_2, \dots, x_ℓ be the children of x , listed in the increasing order. We

define α_x and β_x in the following way:

$$\alpha_x = \beta_{x_\ell} \cdot (2k - 1) \cdot \beta_{x_{\ell-1}} \cdot (2k - 1) \cdots (2k - 1) \cdot \beta_{x_1},$$

and

$$\beta_x = (\alpha_x \cdot (2k))^{|A|}.$$

where $|A|$ is the number of states in \mathcal{A} . Finally, we set $\alpha_t = \alpha_r \cdot (2)^\omega$ where r is the root of the tree t . Notice that the highest priority appearing in α_t is at most d (because the height of t is $d/2$), and so $\alpha_t \in \Sigma_d^\omega$.

In Figure 2, we give the pairs (α_x, β_x) for all non-trivial nodes x in the tree t from Figure 1.

PROPOSITION 5.1. *There is an infinite path in graph G_t whose priority projection is the infinite word α_t .*

Proof. We argue—by induction on the structure of the ordered tree t —that for every node x in t , there are two finite paths containing only vertices in G_t that correspond to leaves in the subtree of t rooted at x , and whose priority projections are the words α_x and β_x , respectively. The base case, when x is a leaf, is straightforward. The inductive case for

$$\alpha_x = \beta_{x_\ell} \cdot (2k - 1) \cdot \beta_{x_{\ell-1}} \cdot (2k - 1) \cdots (2k - 1) \cdot \beta_{x_1}$$

when x is a node with children x_1, x_2, \dots, x_ℓ follows routinely from the inductive hypothesis and the definition of the set of edges with priorities in the graph G_t . The claim for $\beta_x = (\alpha_x \cdot (2k))^{|A|}$ can be equally routinely obtained from the result for α_x . Finally, extending the finite path whose priority projection is α_r , where r is the root of tree t , to an infinite path whose priority projection is α_t is straightforward, because every vertex in G_t has a self-loop with priority 2.

Embedding t isomorphically in $\text{tree}_D(\mathcal{A})$. We prove the following claim for every node x in t ; let the depth of x be $d/2 - k$.

CLAIM 5.1. *Let ρ be a finite run of \mathcal{A} not visiting rejecting states (ρ needs not start in an initial state) that either:*

- reads α_x and is such that all states visited by ρ belong to the same equivalence class of \preceq_{2k+1} , or
- reads β_x .

Then there exists a node $x' = \langle e_{d-1}, e_{d-3}, \dots, e_{2k+1} \rangle$ of $\text{tree}_D(\mathcal{A})$ such that

1. the classes $e_{d-1}, e_{d-3}, \dots, e_{2k+1}$ contain some state visited by ρ ,
2. the subtree of t rooted at x embeds isomorphically in the subtree of $\text{tree}_D(\mathcal{A})$ rooted at x' .

Let us first show how this claim finishes the proof of Lemma 5.3. Because $\alpha_t \in \text{EvenCycles}_{n,d}$, there is a run of \mathcal{A} that reads α_t and never visits rejecting states; let ρ be the prefix of this run that reads α_r , where r is the root of t . Recall that the depth of all the leaves in t is $d/2$. All states visited by ρ belong to the same equivalence class of \preceq_{d+1} , because this quasi-order has only one equivalence class. Using the claim for the run ρ , we obtain that tree t embeds isomorphically in $\text{tree}_D(\mathcal{A})$ (note that x' is necessarily the root of $\text{tree}_D(\mathcal{A})$).

We now prove the claim by induction on k . In the base case, when x is a leaf, we have $\alpha_x = \beta_x = \varepsilon$, and as branching directions of the node $x' = \langle e_{d-1}, e_{d-3}, \dots, e_1 \rangle$ we take the equivalence classes of $\preceq_{d-1}, \preceq_{d-3}, \dots, \preceq_1$ containing the only state of ρ ; such a node indeed fulfils Conditions 1. and 2.

We now focus on the inductive step. Suppose first that ρ reads α_x , and that all visited states belong to the same equivalence class of \preceq_{2k+1} . Let x_1, x_2, \dots, x_ℓ be the children of x . Recall that

$$\alpha_x = \beta_{x_\ell} \cdot (2k-1) \cdot \beta_{x_{\ell-1}} \cdot (2k-1) \cdots (2k-1) \cdot \beta_{x_1}$$

We can divide ρ into fragments $\rho_\ell, \rho_{\ell-1}, \dots, \rho_1$, where ρ_i reads β_{x_i} . For every $i \in \{1, 2, \dots, \ell\}$, by the inductive hypothesis, we can find a node $x'_i = \langle e_{i,d-1}, e_{i,d-3}, \dots, e_{i,2k-1} \rangle$ that fulfils Conditions 1. and 2. with respect to ρ_i and x_i . By assumption, all states visited by ρ belong to the same equivalence class of \preceq_{2k+1} , hence also to the same equivalence class of \preceq_j for all $j \in \{2k+1, 2k+3, \dots, d-1\}$. On the other hand, by Condition 2., the classes $e_{i,j}$ for $i \in \{1, 2, \dots, \ell\}$ and $j \in \{2k+1, 2k+3, \dots, d-1\}$ contain some state visited by ρ . It follows that $e_{i,j} = e_{i',j}$ for all $i, i' \in \{1, 2, \dots, \ell\}$ and $j \in \{2k+1, 2k+3, \dots, d-1\}$, which implies that the nodes $x'_1, x'_2, \dots, x'_\ell$ are siblings, having a common

parent x' . It is easy to see that the node x' satisfies Conditions 1. and 2.

Suppose now that ρ reads $\beta_x = (\alpha_x \cdot (2k))^{|\mathcal{A}|}$. By the fact that D is a tree-decomposition of \mathcal{A} , we know that no transition in ρ goes up with respect to the quasi-order \preceq_{2k+1} , because all priorities in β_x are at most $2k$. As \mathcal{A} has at most $|\mathcal{A}|$ states, it means that at most $|\mathcal{A}| - 1$ transitions of the considered run cause a decrease with respect to \preceq_{2k+1} . This implies that there is a part of this run that reads α_x and contains no increase nor decrease with respect to \preceq_{2k+1} , that is, all states visited by this part of the run belong to the same equivalence class of \preceq_{2k+1} . We continue with this part of the run as in the previous paragraph, and in this way we finish the proof of the claim.

6 Open questions and further work

We leave open the question whether a stronger version of our main technical result holds, namely whether every safety automaton separating EvenCycles_n from OddCycles_n has at least quasi-polynomial number of states. Our argument cannot be directly extended to that setting, as already the proof of Lemma 5.2 crucially relies on that fact that no word from LimsupOdd is accepted.

We also leave it as open questions whether there are parity automata as in Theorem 4.3 but with a sub-quasi-polynomial number of states and a logarithmic number of priorities, or whether quasi-polynomial lower bounds can also be established for such parity automata.

A very recent technical report by Colcombet and Fijalkow [7] gives alternative proofs of our Theorem 4.1 and Lemma 5.3. They do so by introducing a new concept of *universal graphs*, and they argue that a subclass of universal graphs that satisfies an extra (“saturation”) property corresponds to universal trees. Their main result is that the sizes of the smallest strong separators, universal trees, and universal graphs are equal to one another, which is a refinement of the corollary of our Theorem 4.1 and Lemma 5.3 that the former two are. However, in order to conclude that all those smallest sizes are quasi-polynomial, they too rely on our Theorem 2.3.

Acknowledgements

This research has been supported by the EPSRC grant EP/P020992/1 (Solving Parity Games in Theory and Practice). W. Czerwiński and P. Parys are partially supported by the Polish National Science Centre grant 2016/21/D/ST6/01376. N. Fijalkow is supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1 and the DeLTA project (ANR-16-CE40-0007).

References

- [1] J. Bernet, D. Janin, and I. Walukiewicz. Permissive strategies: from parity games to safety games. *Informatique Théorique et Applications*, 36(3):261–275, 2002.
- [2] H. Björklund and S. G. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210–229, 2007.
- [3] M. Bojańczyk and W. Czerwiński. An automata toolbox, February 2018. <https://www.mimuw.edu.pl/~bojan/papers/toolbox-reduced-feb6.pdf>.
- [4] A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theor. Comput. Sci.*, 178(1–2):237–255, 1997.
- [5] C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *STOC*, pages 252–263, 2017.
- [6] K. Chatterjee, M. Henzinger, and V. Loitzenbauer. Improved algorithms for parity and Streett objectives. *Logical Methods in Computer Science*, 13(3), 2017.
- [7] T. Colcombet and N. Fijalkow. Parity games and universal graphs. *CoRR*, abs/1810.05106, 2018. <https://arxiv.org/abs/1810.05106>.
- [8] A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.
- [9] C. Daskalakis and Ch. Papadimitriou. Continuous local search. In *SODA*, pages 790–804, 2011.
- [10] C. Daskalakis, Ch. Tzamos, and M. Zampetakis. A converse to Banach’s fixed point theorem and its CLS completeness. In *STOC*, pages 44–50, 2018.
- [11] L. Daviaud, M. Jurdziński, and M. Lazić. A pseudo-quasi-polynomial algorithm for solving mean-payoff parity games. In *LICS*, pages 325–334, 2018.
- [12] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *FOCS*, pages 368–377, 1991.
- [13] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of μ -calculus. In *CAV*, pages 385–396, 1993.
- [14] E. A. Emerson and Ch.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus (Extended abstract). In *LICS*, pages 267–278, 1986.
- [15] J. Fearnley. Exponential lower bounds for policy iteration. In *ICALP*, pages 551–562, 2010.
- [16] J. Fearnley, S. Jain, S. Schewe, F. Stephan, and D. Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In *SPIN*, pages 112–121, 2017.
- [17] N. Fijalkow. An optimal value iteration algorithm for parity games. *CoRR*, abs/1801.09618, 2018. <https://arxiv.org/abs/1801.09618>.
- [18] O. Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *LICS*, pages 145–156, 2009.
- [19] O. Friedmann, T. D. Hansen, and U. Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In *STOC*, pages 283–292, 2011.
- [20] H. Gimbert and R. Ibsen-Jensen. A short proof of correctness of the quasi-polynomial time algorithm for parity games. *CoRR*, abs/1702.01953, 2017. <https://arxiv.org/abs/1702.01953>.
- [21] M. Goldberg and E. Lifshitz. On minimal universal trees. *Matematicheskie Zametki*, 4(3):371–380, 1968. (In Russian).
- [22] T. A. Henzinger and N. Piterman. Solving games without determinization. In *CSL*, pages 395–410, 2006.
- [23] M. Jurdziński. Deciding the winner in parity games is in $\text{UP} \cap \text{co-UP}$. *Inf. Process. Lett.*, 68(3):119–124, 1998.
- [24] M. Jurdziński. Small progress measures for solving parity games. In *STACS*, pages 290–301, 2000.
- [25] M. Jurdziński and R. Lazić. Succinct progress measures for solving parity games. In *LICS*, pages 1–9, 2017.
- [26] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. Comp.*, 38(4):1519–1532, 2008.
- [27] K. Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *LICS*, pages 639–648, 2018.
- [28] A. W. Mostowski. Games with forbidden positions. Technical Report 78, Uniwersytet Gdański, 1991.
- [29] M. O. Rabin. *Automata on Infinite Objects and Church’s Problem*. American Mathematical Society, 1972.
- [30] S. Schewe. Solving parity games in big steps. *J. Comput. Syst. Sci.*, 84:243–262, 2017.
- [31] H. Seidl. Fast and simple nested fixpoints. *Inf. Process. Lett.*, 59(6):303–308, 1996.
- [32] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *CAV*, pages 202–215, 2000.
- [33] U. Zwick and M. Paterson. The complexity of mean-payoff games on graphs. *Theor. Comput. Sci.*, 158:343–359, 1996.