

REGISTER GAMES *

UDI BOKER AND KAROLIINA LEHTINEN

Interdisciplinary Center (IDC) Herzliya, Israel
e-mail address: udiboker@idc.ac.il

University of Liverpool, United Kingdom
e-mail address: k.lehtinen@liverpool.ac.uk

ABSTRACT. The complexity of parity games is a long standing open problem that saw a major breakthrough in 2017 when two quasi-polynomial algorithms were published.

The present article presents a third, independent approach to solving parity games in quasi-polynomial time, based on the notion of *register game*, a parameterised variant of parity games. The analysis of register games not only leads to a quasi-polynomial algorithm for parity games, and a polynomial algorithm for restricted classes of parity games; it also enables a translation of alternating parity word automata into alternating weak automata with a quasi-polynomial blow-up—this improves on the previous exponential translation.

We also use register games to investigate the parity index hierarchy: while for words the index hierarchy of alternating parity automata collapses to the weak level, and for trees it is strict, for structures between trees and words, it collapses *logarithmically*, in the sense that any parity tree automaton of size n is equivalent, on these particular classes of structures, to an automaton with a number of priorities logarithmic in n .

1. INTRODUCTION

A play in a parity game consists of a player, whom we shall call Eve, and her opponent, Adam, moving a token along the edges of a graph labelled with integer priorities, forever, thus forming an infinite path. Eve’s objective is to force the highest priority that occurs infinitely often to be even, while Adam tries to stop her.

These games arise at the intersection of logic, games, and automata theory. In particular, they are the acceptance games for alternating parity automata, both on trees and on ω -words. The complexity of solving parity games—that is, of deciding which player has a winning strategy—still is, despite extended efforts, an open problem: it is in $\text{UP} \cap \text{coUP}$ [Jur98] yet it is not known to admit a polynomial algorithm. After over twenty-five years of incremental improvements, Calude, Jain, Khossainov, Li, and Stephan published the first quasi-polynomial solution [CJK⁺17]. Only a little later in the same year, Jurdziński and

Key words and phrases: Parity games, Alternating automata, Parity automata, Weak automata.

* The article extends [Leh18] and parts of [BL18].

Research supported by the Israel Science Foundation grant 1373/16 and the EPSRC grant EP/P020992/1 (Solving Parity Games in Theory and Practice).

Lazić presented an independent progress-measure based algorithm that achieves the same complexity [JL17]. This article presents a third, independent approach to solving parity games in quasi-polynomial time. This automata-theoretic method enables us to also solve the more general problem of translating alternating parity word automata into alternating weak automata, and offers some new insights on the descriptive complexity of parity games and the parity index problem.

Register Games. Our first contribution is to present *register games*, a parameterised variant of parity games, that we then use to analyse the complexity of parity games and parity automata, both on infinite words and infinite trees. A register game consists of a normal parity game, augmented with a fixed number of registers that keep partial record of the history of the game. Although the register game is harder for Eve than the parity game on the same arena, if she can win the parity game, then she can also win the register-game as long as she has a large enough number of registers. Exactly how many register she needs depends on the parity game arena. We call this the register-index of the parity game: it is a measure of complexity which takes into account both the priority assignment of the parity game and the structure of the underlying graph.

Two key properties of register-games then enable us to derive complexity results for finite parity games as well as parity automata on words and restricted classes of trees. First, register-games are *automata-definable*: there is, for every integer k , an alternating parity automaton that accepts infinite parity games in which Eve wins the k register game; furthermore, for any alternating parity automaton \mathcal{A} , we can define a family of parameterised automata \mathcal{A}_k that use the k -register game as their acceptance game instead of standard parity games. Second, the register-index is bounded *logarithmically* in the number of disjoint cycles of a parity game arena.

A quasi-polynomial algorithm for parity games. For finite arenas, where the number of disjoint cycles is bounded by the number of positions, solving parity games reduces to solving the register game with a number of registers logarithmic in the size of the game. This results in a quasi-polynomial parity game algorithm, as well as a parameterised polynomial algorithm that solves classes of parity games with bounded register-index.

Word automata transformations. The complexity of solving parity games is intimately related to the complexity of turning alternating parity word automata (APW) into alternating weak word automata (AWW). Indeed, solving parity games amounts to checking the emptiness of an APW on the trivial singleton alphabet; the emptiness of AWW on the singleton alphabet can be checked in linear time [KV98]. Hence a translation from APW to AWW immediately yields an algorithm for solving parity games of which the time complexity matches the size-increase of the automata translation. Note, however, that the automata-translation question is more general, since automata need not be defined over a one-letter alphabet, and often a binary, or larger, alphabet adds substantial complexity.

Nevertheless, until 2017, the best known algorithms for the two problems were roughly the same: exponential in the number of priorities. A competitive tool for solving parity games is even based on the translation from parity to weak automata [SMPV16]. In 2017, however, the advent of quasi-polynomial algorithms created a gap between the complexity of solving parity games, and the automata translation.

In this article, we show that the analysis of parity games that are infinite, or at least of unbounded size, allows us to generalise the quasi-polynomial time complexity of solving parity games to the blow-up incurred when turning alternating parity word automata into alternating weak automata.

The index hierarchy. While the translation of alternating parity automata into weak is always possible for word automata [KV01], this is not the case for tree automata. Indeed, while alternating weak automata suffice to capture all ω -regular word languages, no fixed number of priorities suffices to capture all regular tree languages—this follows from the equivalence between alternating parity automata and the modal μ calculus [Wil01], and the strictness of the modal μ alternation hierarchy [Bra98]. We say that the *parity index hierarchy* is strict on trees, but collapses to the weak level over words.

We study automata on structures that are, in some sense, *between* words and trees and show, using register games, a *logarithmic* collapse of the index hierarchy: for every alternating parity automaton \mathcal{A} with n states, there is an alternating parity automaton \mathcal{A}' with only $O(\log n)$ priorities that is equivalent to \mathcal{A} over these structures.

Of other quasi-polynomial automata. We conclude this article with a discussion of how existing quasi-polynomial parity game algorithms, and the automata that can be derived from them, differ. In particular, building on Bojańczyk and Czerwiński [BC18]’s discussion of Calude et al’s and Jurdziński and Lazić’s algorithms as good-for-games word automata, we discuss how they can also be seen as parity automata on trees and argue that the automaton for register games on the other hand is not good-for-games. We then identify a sufficient condition for an alternating parity tree automaton to induce a translation from alternating parity word automata to alternating weak automata.

This article is based on Lehtinen’s quasi-polynomial algorithm for parity games [Leh18], which introduced the notion of register games, and Boker and Lehtinen’s extension of this technique into a translation of parity word automata into weak automata [BL18]. Here the definition of register-games is simplified, generalised to infinite parity games, and presented from an automata-theoretic, rather than μ -calculus, perspective.

2. PARITY GAMES

Definition 2.1 (Parity games). A parity game is an infinite-duration two-player zero-sum path-forming game, played between Eve and her opponent Adam on a potentially infinite game graph $G = (V, V_E, V_A, E, \Omega)$ called the *arena*. The *positions* V of the arena are partitioned into those belonging to Eve, V_E , and those belonging to Adam, V_A . The *priority assignment* $\Omega : V \rightarrow I$ maps every position in V to a *priority* in a finite co-domain $I = \{i, \dots, d\}$ where $i \in \{0, 1\}$. The edge-relation $E \subseteq V \times V$ defines the successors of each position. Without loss of generality we assume all positions to have at least one successor.

A play is an infinite sequence of positions $\pi = v_0 v_2 \dots$ such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$. A play is winning for Eve if the highest priority that occurs infinitely often along π is even; otherwise it is winning for Adam.

A (positional) *strategy* σ for a player $P \in \{\text{Adam}, \text{Eve}\}$ in a parity game G maps every position v belonging to P in G to one of its successors. A play is said to agree with a strategy σ for P if $v_{i+1} = \sigma(v_i)$ whenever v_i belongs to P . A strategy σ for player P is

said to be winning for P from a position v if all plays starting at v that agree with σ are winning for P . We call the positions from which Eve has a winning strategy Eve's winning region in G , written $W_E(G)$; Adam's winning region is written $W_A(G)$.

We write G, v for the parity game G with a designated initial position v . A winning strategy in G, v is a strategy that is winning from v .

Theorem 2.2 (Positional Determinacy [EJ91, Mos91]). *In all positions in a parity game, one of the players has a positional winning strategy.*

It will sometimes be convenient, for clarity and aesthetics, to assign priorities to edges instead of vertices, with $\Omega : E \rightarrow I$. A parity game with edge priorities can be converted into one with node priorities by introducing intermediate, priority-carrying nodes onto edges and assigning a low priority to other vertices. Conversely, a vertex-labelled parity game can be converted into an edge-labelled one by assigning the priority of a node to its outgoing edges.

3. REGISTER GAMES

This section describes the key technical development of this article: register games. These are parameterised variations of parity games, also played on a parity game arena. Crucially, the winning condition of a k -register game is a parity condition that ranges over priorities $[0..2k + 1]$ rather than the priorities of the arena. The larger the parameter k , the easier the k -register game becomes for Eve who, on arenas in which she has a winning *parity game* strategy, is guaranteed to also have a winning k -register game strategy, for some large enough k . Note that the mechanics of register games have been simplified, compared to their first appearance in [Leh18]; see Remark 3.5.

3.1. Definitions and Observations. Informally, the k -register game consists of a normal parity game, augmented with a tuple (r_0, \dots, r_k) of registers that keeps a partial record of the history of the game. During a turn, several things happen: the player whose turn it is in the parity game moves onto a successor position of their choice, which has some priority p and Eve chooses an index i , $0 \leq i \leq k$; then, the registers get updated according to both i and p , and an output between 0 and $2k + 1$ is produced, also according to i and p .

The update wipes out the contents of registers with index lower than i : for $j < i$, r_j is set to 0. Meanwhile r_i is set to p and r_j for $j > i$ to $\max(r_j, p)$. If $i > 0$, the output is $2i$ if $\max(r_i, p)$ is even, and $2i + 1$ otherwise. Then, in the limit, Eve wins a play if the largest output that occurs infinitely often is even.

Since the winning condition of the register game is a parity condition, we can formally define the k -register game on a parity game arena G as a parity game on an arena $\mathcal{R}_E^k(G)$, of which the positions are positions of G paired with vectors in I^{k+1} that represent the contents of the registers. An additional binary variable t indicates whether the next move consists of Eve's choice of register ($t = 0$), or a move in the underlying parity game ($t = 1$).

Definition 3.1 (Register game). Let G be a parity game $(V^G, V_E^G, V_A^G, E^G, \Omega^G)$ and let I be the co-domain of $\Omega^G : V^G \rightarrow I$. For a fixed parameter $k \in \mathbb{N}$, the arena of the k -register game $\mathcal{R}_E^k(G)$ on G in which Eve controls the registers, consists of $\mathcal{R}_E^k(G) = (V, V_E, V_A, E, \Omega)$ as follows.

While G carries its priorities on its vertices, for the sake of clarity, $\mathcal{R}_E^k(G)$ carries them on its edges, $\Omega : E \rightarrow [0..2k + 1]$.

- V is a set of positions $(v, \bar{r}, t) \in V^G \times I^{k+1} \times \{0, 1\}$,
- V_A consists of $(v, \bar{r}, 1)$ such that $v \in V_A^G$,
- V_E consists of $V \setminus V_A$,
- E is the disjoint union of sets of edges E_{move} and E_i for all $i \in [0..k]$ where:
 - E_{move} consists of edges $((v, \bar{r}, 1), (w, \bar{r}, 0))$ such that $(v, w) \in E^G$.
 - For each $i \in [0..k]$, E_i consists of edges $((v, \bar{r}, 0), (v, \bar{r}', 1))$ such that:
 - $r'_j = \max(r_j, \Omega^G(v))$ for $j > i$,
 - $r'_j = \Omega^G(v)$ for $j = i$, and
 - $r'_j = 0$ for $j < i$.
- Ω assigns priorities from $[0..2k+1]$ to edges as follows:
 - Edges of E_{move} have priority 0;
 - $((p, \bar{r}, 0), (p, \bar{r}', 1)) \in E_i$ has priority $2i$ if $\max(r_i, p)$ is even, and priority $2i+1$ otherwise.

Terminology. Given a play in $\mathcal{R}_E^k(G)$, we call the *underlying play* its projection onto the first element of each visited position. At a position (v, \bar{r}, t) , we write that: a priority $p \in I$ *occurs* if $\Omega^G(v) = p$; a register $i \in [0..k]$ *contains* a priority $p \in I$ if $r_i = p$; Eve *chooses* register i and *outputs* $j \in [0..2k+1]$ if the play follows an edge in E_i of priority j .

A strategy for Adam in G induces a strategy for Adam in $\mathcal{R}_E^k(G)$. A strategy for Eve in G paired with a register-choosing strategy in $\mathcal{R}_E^k(G)$ induces a strategy for Eve in $\mathcal{R}_E^k(G)$. Observe that because the winning condition depends on the outputs which occur infinitely often, and that registers up to the largest one chosen infinitely often renew their contents infinitely often, if a player has a winning strategy in $\mathcal{R}_E^k(G)$ from (v, \bar{r}, t) , then they have a winning strategy from all $(v, -, -)$. We will then simply say that they have a winning strategy from v .

The k -register game arena $\mathcal{R}_A^k(G)$ where Adam controls the registers is similar to $\mathcal{R}_E^k(G)$ except that positions $(v, \bar{r}, 0)$ are in V_A , edges in E_{move} have priority 0, and edges $((v, \bar{r}, 0), (v, \bar{r}', 1))$ of E_i have priority $2i+2$ if r_i is even, and $2i+1$ otherwise. The k -register game with Eve (resp., Adam) in control of registers on a parity game arena G is the parity game on the arena $\mathcal{R}_E^k(G)$ (resp., $\mathcal{R}_A^k(G)$). Unless specified, the k -register game on G refers to $\mathcal{R}_E^k(G)$.

For fixed k , $\mathcal{R}_E^k(G)$ is a parity game on an arena of size polynomial in the size of G and of priority domain $[0..2k+1]$: it can be solved in polynomial time in the size of G [Jur00].

We now establish two important facts about register-games: if Adam has a winning strategy from a position v in a parity game G , then he has a winning strategy from positions (v, \bar{r}, t) in $\mathcal{R}_E^k(G)$ for any k . However, if Eve has a winning strategy in G from v , then she also has a winning strategy from (v, \bar{r}, t) in $\mathcal{R}_E^k(G)$ as long as k is taken to be large enough.

Lemma 3.2. *If Adam has a winning strategy in the parity game G at v , then he also has a winning strategy in $\mathcal{R}_E^k(G)$ starting at v for all k .*

Proof. Assume Adam has a strategy τ in G that is winning from v . On any play π starting at v that agrees with τ , the highest priority p seen infinitely often is odd. Let i be the highest register chosen infinitely often by Eve during π . Then, eventually—that is, after the last occurrence of anything higher than p and after Eve no longer chooses registers

higher than i —whenever p is seen, r_i is set to p and remains at p until Eve again chooses i . Then, since p is odd, this outputs $2i + 1$. Since p occurs infinitely often and i is picked infinitely often, $2i + 1$ is output infinitely often. It is also the highest value output infinitely often because i is the highest index picked infinitely often. Since $2i + 1$ is odd, π is winning for Adam. τ is therefore a winning strategy for Adam in $\mathcal{R}_E^k(G)$ from position v . \square

Lemma 3.3. *If Eve has a winning strategy in the parity game G with parity co-domain I at a position v_i , then she also has a winning strategy in $\mathcal{R}_E^k(G)$ from v_i for $k \geq i$ where $2i$ is the largest even priority in I .*

Proof. Given a strategy σ in the parity game G that is winning for Eve from v_i , let σ' be the following strategy for Eve in $\mathcal{R}_E^k(G)$ where $k \geq i$ and $2i$ is the largest even priority in I : at positions $(v, \bar{r}, 1)$ the strategy σ' follows σ , that is if v belongs to Eve, then $\sigma'(v, \bar{r}, 1) = (w, \bar{r}, 0)$ where $\sigma(v) = w$; at positions $(v, \bar{r}, 0)$ where $\Omega^G(v) = 2i$ or $2i + 1$ for some i , the strategy σ' chooses register i .

We now argue that σ' is winning for Eve in $\mathcal{R}_E^k(G)$ from v_i . Since σ' follows σ in the underlying game, the highest priority p that occurs on any play beginning at v_i that agrees with σ' is even, say $p = 2i$ for some i . The highest register chosen infinitely often is therefore i . Since eventually nothing higher than p occurs anymore, r_i will eventually remain p in perpetuity, and therefore, eventually, every time Eve chooses i , this outputs $2i$; since this is the highest register chosen infinitely often, nothing higher is output infinitely often. Hence Eve wins every play that agrees with σ' from v_i . \square

The number of registers that Eve needs to win the register-game on a parity game in which she has a winning strategy depends on the complexity of her winning strategy. We define this as the *register index* of a parity game and consider it as a measure of complexity for parity games, comparable to measures such as entanglement [BGKR12].

Definition 3.4 (Register-index). A parity game G has register-index k at $v \in W_E(G)$ if Eve wins $\mathcal{R}_E^k(G)$ from v and at $v \in W_A(G)$ if Adam wins $\mathcal{R}_A^k(G)$ from v . A parity game G has register index k if it has register index up to k at all positions.

Remark 3.5. The register game defined here differs slightly from the one defined in [Leh18] and used in [BL18]. Here, for a more elegant presentation, Eve has to choose a register at every turn, but has an additional 0-indexed register she can default to. This avoids having to use an additional low priority to encode the requirement that Eve resets infinitely often. This also means that parity games in which Eve has a strategy that can avoid seeing odd priorities infinitely often have register-index 0. Furthermore, and perhaps more significantly, the register update mechanism is simplified: rather than a cumbersome shift of the values in the registers, here all registers below the chosen one simply get reset to 0 while the chosen register gets the current priority. This new game mechanism allows for simpler strategies; in particular the proof of Theorem 4.2 is now somewhat simpler.

Finally, here we do not restrict register-games to finite arenas, but consider them on potentially infinite ones.

3.2. Examples. In this section we explore which features of parity games affect the register index, and which do not. Since the register-index depends on a player's winning regions, the examples presented in this section are all on one-player games: all positions belong to Adam. They can of course be embedded into two-player games of arbitrary complexity;

however, for the register-index, only the game induced by the simplest winning strategy matters.

We begin by looking at register games with a small number of registers, starting with 0-register games. We observe that Eve wins the 0-register game if she has a strategy σ such that any play that agrees with σ sees finitely many odd priorities, such as the game depicted in Figure 1. Indeed, if Eve follows such a strategy, then eventually all outputs in the 0-register games are 0. Hence register-index 0 characterizes particularly simple games in which Eve has a winning strategy for which odd priorities occur only a finite (but unbounded) number of times on any play. Among well-known parity games with register-index 0 are the example games for which strategy improvement and divide-and-conquer algorithms exhibit worst-case complexity [BDM17, Fri09].

While register-index 0 indicates that Eve’s winning strategy must be extremely simple, register-index 1 already captures parity games with more complexity. Example 3.6 for instance discusses games with high entanglement, tree-width and a large number of priorities that have register-index 1. Known examples of parity games of register-index 1 are the families of parity games that exhibit worst-case complexity for Zielonka’s recursive algorithm and the quasi-polynomial progress measure algorithms [Fri11, FJS⁺17]. In these games every odd priority is immediately followed by a larger even priority, so Eve still has an easy 1-register game strategy consisting of choosing register 1 whenever an even priority occurs, and register 0 otherwise. We also note that sequences of duplicate priorities do not affect the register-index since Eve can choose register 0 on all but one of the priorities in the sequence; this means that parity games with sequences of duplicate priorities—which can also be used to force progress measure algorithms to exhibit high complexity—do not have high register-index.

Example 3.6. Figure 2 shows an edge-labelled arena in which Eve wins the parity game but loses the 0-register game. In the 0-register game, Adam’s strategy is to loop at the current position once (this clears the register), then move to the other position and repeat; Eve has no choice but to produce outputs 1 and 0 infinitely often. Eve can win the 1-register game by choosing register 1 after seeing 2, and register 0 after seeing 1 or 0.

Figure 3 illustrates a slightly more complicated family of parity games, which has linear entanglement, tree-width and number of priorities, yet constant register-index 1. Eve’s strategy is to choose register 1 whenever she sees an even priority and register 0 otherwise. Since odd priorities only occur after a larger even priority, with this strategy register 1 permanently contains even priorities. Then every occurrence of an even priority leads to output 2, while odd priorities lead to output 1.

We have now seen various parity games of low register-index. Building parity games with high register-index is trickier, and since the structures of these games are more involved, it is no surprise that they don’t seem to appear in the literature.

Lemma 3.7. *For all n , there exists a parity game of register-index at least n .*

Proof. Let H_0 be the game arena consisting of a single node, belonging to Adam, with a self-loop of priority 0. This unique node is also the initial node of H_0 .

Then, for all $n > 0$, the arena H_n consists of two distinct copies of H_{n-1} with initial positions v_0 and v_1 respectively, with an edge (v_0, v_1) of priority $2n - 1$ and an edge (v_1, v_0) of priority $2n$. The position v_0 is also the initial position of H_n . See Figure 4.

Eve wins these parity games H_n because all cycles are dominated by an even priority. We will show that for $n > 0$, Adam has a winning strategy in the $n - 1$ -register game

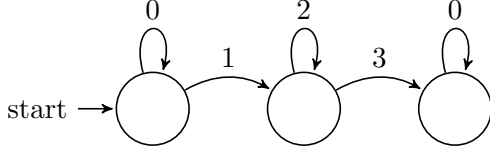


Figure 1: Parity game of register-index 0

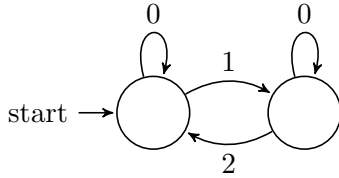


Figure 2: Parity game of register-index 1

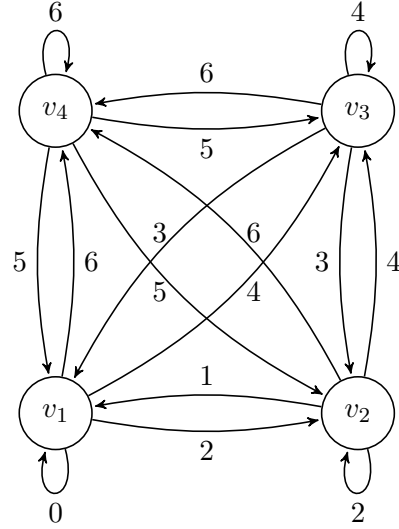


Figure 3: Parity game of register-index 1

$\mathcal{R}_E^{n-1}(H_n)$.

We reason inductively, and show that for each H_n , $n > 0$, that i) from the initial position in $\mathcal{R}_E^m(H_n)$ for $m \geq n - 1$, with a register configuration in which register contents are bounded by $2n - 1$, Adam can force the game to output $2n - 1$ or a higher odd priority, before returning to the initial position, and ii) Eve loses in $\mathcal{R}_E^{n-1}(H_n)$.

Base case: Adam has a winning strategy in $\mathcal{R}_E^0(H_1)$: His strategy is to loop once in the current position to see 0—this sets the register content to 0; he then moves to the other position and repeats. This causes both 0 and 1 to be output infinitely often.

Note that if Adam uses this strategy in $\mathcal{R}_E^m(H_1)$ for $m \geq 0$, starting from a register configuration in which register contents are bounded by 1, although he can't necessarily win, he can force the game to output 1 or a higher odd priority before returning to the initial position.

Inductive step: Assume i) and ii) for H_n .

i) Consider the following strategy for Adam in $\mathcal{R}_E^m(H_{n+1})$ for $m \geq n$. He first moves from v , the initial position of H_{n+1} onto the initial position v' of the second component of H_{n+1} , via the odd priority $2n + 1$. Then, he plays in H_n , which only contains priorities smaller than $2n + 1$, with a strategy that is winning in $\mathcal{R}_E^{n-1}(H_n)$. To counter this strategy, Eve has to eventually choose a register of index n or higher, after which Adam returns to the initial position. This is his strategy τ_n . Observe that if at the initial position all register contents are bounded by $2n + 1$, then Eve will either lose in the second H_n component, or choose a register of index n or higher when it contains the odd priority $2n + 1$, outputting $2n + 1$ or a higher odd priority.

ii) We now show that he also has a winning strategy in $\mathcal{R}_E^n(H_{n+1})$. He begins by playing τ_n until $2n + 1$ is output and the play is back at the initial position. Note that some registers now might contain $2n + 2$, so he can not yet repeat τ_n . Instead, he plays a strategy that is winning in $\mathcal{R}_E^{n-1}(H_n)$ in the *first* H_n component

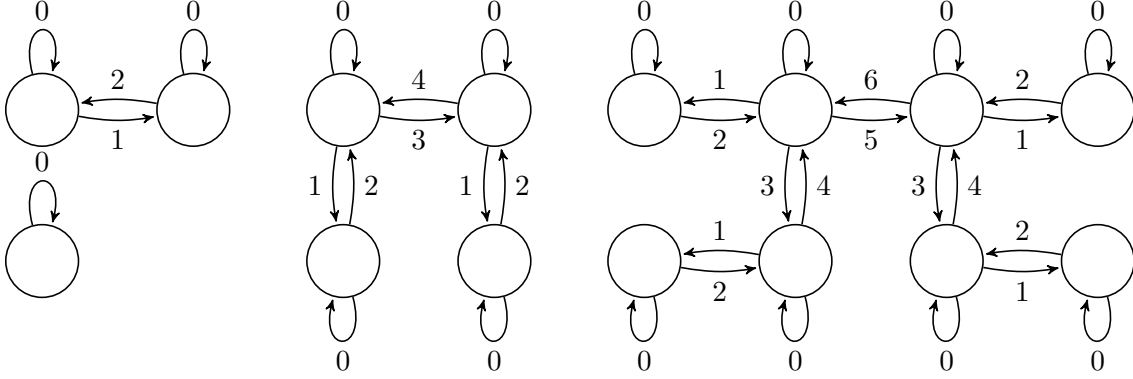


Figure 4: H_0, H_1, H_2, H_3 : A family of parity games of high register-index

of H_{n+1} . Again, Eve will lose unless she chooses register n . After she has chosen the register n , all the registers hold values smaller than $2n + 1$. Adam can then return to the initial position, and again use τ_n to force output $2n + 1$. Thus by alternating between using τ_n to force the maximal odd output, and clearing the registers of higher priorities in order to be able to use τ_{n+1} again, Adam forces $2n + 3$ to be output infinitely often.

Hence H_n has register-index at least n .

□

3.3. The Register-index as a measure of complexity. Given the elusiveness of a polynomial algorithm for parity games, there is a rich line of research in algorithms that are polynomial for restricted classes of parity games. On one hand, the classical parity game algorithms, which are exponential in the number of priorities, are polynomial on parity games with a fixed number of priorities. In this sense, the size of the priority co-domain is perhaps the simplest way of measuring the complexity of parity games. On the other hand, there are many graph-theoretic restrictions, such as bounded tree-width, bounded clique-width and bounded entanglement, which also allow for polynomial algorithms. These two classes of restrictions are orthogonal in the sense that the size of the priority assignment is agnostic to the underlying graph and vice-versa.

In contrast, the register-index can be seen as a measure of complexity that takes into account both the complexity of the priority assignment and the structure of the underlying parity game graph. Since for bounded k the register game over a game G is just a parity game of polynomial size in $|G|$, solving parity games of bounded register-index is polynomial. Since we may not know the register-index in advance, by solving both $\mathcal{R}_E^k(G)$ and $\mathcal{R}_A^k(G)$ up to fixed k , we obtain a parameterised polynomial algorithm which solves games of register-index up to k , and does not return an answer for other arenas.

From the analysis in the previous sections, this algorithm seems likely to be effective on many—perhaps most—reasonable parity games. In particular, parameter 1 suffices to solve the various parity games that are hard for existing algorithms, including the quasi-polynomial progress measure algorithm; furthermore, the register-index is independent of

measures such as tree-width and entanglement. In this sense, this algorithm is complementary to existing ones. In any case, as we shall see in the next section, a logarithmic parameter suffices to solve *all* finite parity games, making this algorithm quasi-polynomial.

4. REGISTER GAMES AND FINITE PARITY GAMES

So far we have defined both parity and register games on potentially infinite arenas. In this section we consider the special case of finite arenas, which is of particular interest for verification and synthesis. We first show that for finite parity games, the register-index is logarithmically bounded in the number of disjoint cycles, and by extension in the number of states of the game. We deduce a quasi-polynomial algorithm for solving finite parity games.

4.1. Logarithmic bound on the register index of finite parity games. This section presents the key technical result, from which the various results in the sequel follow: the register-index is logarithmic in the number of disjoint cycles in a finite parity game.

For a finite directed graph G , we call *dc-size* the maximal number of vertex-disjoint cycles in G ; the dc-size of a parity game is that of the underlying directed graph.

To better build strategies inductively from strategies over subgames, we define the notion of *defensive* strategy. Indeed, an arbitrary winning strategy in a subgame may output a finite number of large odd priorities; this is problematic when this strategy is used in a subgame that a play can enter infinitely often. Defensive strategies avoid high odd priorities from the start.

Definition 4.1 (Defensive register-index). For a subgame G with maximal priority d , in which Eve has a winning strategy, a winning strategy σ for Eve in $\mathcal{R}_E^k(G)$ is *defensive* if, from positions $(v, \bar{r}, 0)$ where $r_k \geq d$ and r_k is even, a play that agrees with σ never outputs $2k + 1$. G has defensive register-index k if Eve has a defensive winning strategy in $\mathcal{R}_E^k(G)$ from all positions.

Theorem 4.2. *The register-index k of a finite parity game of dc-size z is at most $1 + \log z$.*

Proof. From the definition of register-index, it suffices to consider the single-player parity games G induced by any winning strategy for Eve in her winning region. Observe that in the register games on G , all positions with multiple successors belong to Adam and Eve's strategy consists of just choosing a register after each move in the underlying game.

We show by induction on the number of positions n in G that the *defensive* register-index of G is bounded by $1 + \log z$; the theorem follows. The base case, $n = 1$, is trivial. For the inductive step where G has at least two states, let G' be the game induced by positions of G of priority up to $p - 2$, where p is the maximal even priority that appears in some cycle of G . Then, let G_1, \dots, G_j be the *maximal strongly connected subgames* of G' . Let k_1, \dots, k_j be their respective defensive register-indices, and m the maximal among these. Observe that $j \leq z$. If there are no such subgames, then all cycles in G contain p in which case Eve wins defensively in $\mathcal{R}_E^1(G)$ by choosing register 1 when p occurs in a cycle and register 0 otherwise.

Case of a unique i for which $k_i = m$ and $m > 0$: We show that the defensive register-index of G is no more than m . Since the dc-size of G_i is no larger than z , and from the inductive hypothesis $m \leq 1 + \log z$, this suffices.

Eve's strategy in the m -register game on G is as follows: within a subgame G_j she uses the bottom ranking k_j registers to simulate her defensive winning strategy in the k_j -register game on G_j ; elsewhere, she chooses register m whenever p occurs in a cycle and register 0 otherwise.

Assume that this strategy is played from a register-configuration where the top register contains an even priority greater or equal to the largest priority in G . First observe that after choosing register m upon seeing p in a cycle, r_m is set to p . Furthermore, since this strategy encounters p between any two entrances into G_i , and Eve chooses register $0 \neq m$ when p does not occur, it always enters G_i with either p or the larger even initial r_m -value in register m . Thus, since the strategy within G_i is defensive, it never outputs $2m + 1$ in G_i . Similarly, when p occurs in a cycle, $\max(r_m, p)$ is always even: r_m is either its initial value, p , or a smaller priority from G_i . Adam then has the choice of eventually staying within a subgame G_j where Eve follows a winning strategy, or changing subgames infinitely often. In the latter case, p is seen infinitely often in a cycle, thus producing $2k_m$ as output infinitely often. This strategy is therefore both defensive and winning for Eve.

Note that if this strategy is played from a register-configuration in which the top register is not an even priority greater or equal to the highest priority in G , then a play might output $2k_m + 1$ once, but the values output infinitely often will still be the same as above, so the strategy is still winning.

Case of i, j where $i \neq j$ and $k_i = k_j = m$: We show that the defensive register-index of G is no more than $m + 1$. This suffices, since by the induction hypothesis, each of G_i and G_j has dc-size at least 2^{k_m-1} ; then G has dc-size at least 2^{k_m} as G_i and G_j are disjoint.

Eve's strategy in the $m + 1$ -register game on G is as follows: in a subgame G_i , use the registers up to k_i to simulate a winning strategy in the k_i -register game on G ; elsewhere, choose the register $m + 1$ whenever p is seen in a cycle, and register 0 otherwise. This strategy is winning since every play either eventually stays within a subgame where Eve is following a winning strategy, or it enters some subgame infinitely often. In this case, it also must see p occurring in a cycle infinitely often; then $2(m + 1)$ is output infinitely often while nothing higher is ever output infinitely often since after the first occurrence of p in a cycle, register $m + 1$ permanently contains p . Furthermore, if the initial content of register m is an even priority larger than any occurring in G , then every time Eve chooses register m , the output is even; the strategy is therefore defensive.

Case of $m = 0$: Eve can win the 1-register game on G , using a strategy as above: within a subgame G_i , she uses her strategy in the 0-register game, and chooses register 1 whenever p is seen in a cycle. This strategy is winning since a play either remains in a subgame and follows a winning strategy, or sees p infinitely often in a cycle and therefore outputs 2 infinitely often, but does not output 3 infinitely often. If initially the top register contains an even priority greater or equal to the maximal priority in G , this strategy never outputs 3 and is therefore defensive.

□

Corollary 4.3. *The register-index k of a finite parity game of size z is at most $1 + \log z$.*

Remark 4.4. The logarithmic bound was shown in [Leh18] with respect to the size of a finite parity game directly; in [BL18] the logarithmic bound was strengthened to operate with

respect to the so-called *scc-size*, that is, the maximal number of disjoint strongly connected components. Here we opt for the number of vertex-disjoint cycles, which is clearly equal to the scc-size, but perhaps more intuitive.

4.2. A quasi-polynomial algorithm for parity games. The quasi-polynomial solvability of parity games then follows: to solve a parity game G of size n , one can always solve $\mathcal{R}_E^{1+\log n}(G)$ instead.

Corollary 4.5 (Also from [CJK⁺17, JL17]). *Parity games are solvable in quasi-polynomial time.*

Proof. Let n be the number of vertices in a parity game G with d distinct priorities, and m edges. The parity game $\mathcal{R}_E^k(G)$ has $\mathcal{O}(nd^{k+1})$ positions, priorities up to $2k + 1$ and $\mathcal{O}((m + nk)d^k)$ edges. Since $\mathcal{R}_E^k(G)$ is presented with its priorities on its edges, for the complexity analysis we take the size of $\mathcal{R}_E^k(G)$ to be $\mathcal{O}(knd^k)$, to account for an additional vertex for each of the $\mathcal{O}(knd^k)$ edges of significant priority – that is, those in E_r for some $r \in [0..k]$.

The register-index of a parity game of size n is at most $1 + \log n$. Therefore solving parity games G reduces to solving $\mathcal{R}_E^k(G)$ where $k = 1 + \log n$. The $\mathcal{R}_E^k(G)$ game can then be solved with an algorithm exponential in the number of priorities of one’s choice, say the small progress measure algorithm [Jur00], to obtain a quasi-polynomial $2^{\mathcal{O}((\log n)^3)}$ algorithm. \square

Without further optimisations, the complexity of this algorithm is not competitive with respect to existing quasi-polynomial algorithms: the time-complexity is worse by a quasi-polynomial factor, and it is not clear how to avoid the space-complexity involved in building $\mathcal{R}_E^{1+\log n}(G)$. However, as discussed in Section 3.2, many games have very low register-index, including those that exhibit worst-case complexity for other algorithms; on such games this approach is expected to work well. Section 7 discusses further ideas to adapt this approach for practical usage.

Furthermore, One of the principal appeals of this technique for analysing parity games comes undoubtedly from the logic and automata-theoretic perspective. In the sequel, we turn our attention to infinite parity games, or at least parity games of unbounded size, which we use to build automata transformations based on register-games.

5. REGISTER GAMES AND PARITY AUTOMATA

Parity automata are closely related to parity games: on one hand, the language of parity games (represented as infinite trees) with a fixed number of priorities in which Eve has a winning strategy is recognised by an alternating parity tree automaton; on the other hand, the acceptance game for alternating parity automata—both on trees and on words—is an infinite parity game.

In this section we study the automata-theory of register games. We first establish that register-games are *automata-definable* and use this to interpret the quasi-polynomial complexity of parity games in terms of *descriptive complexity*. Note that we are concerned here with the complexity of recognising parity games *with a fixed number of priorities* in which Eve has a winning strategy; for parity games with an unbounded number of priorities, this task is beyond the expressivity of parity automata and the μ calculus [DG08].

Then we use register games on *infinite* and *unbounded* arenas to better understand the trade-offs between different automata acceptance conditions. In particular, we present a quasi-polynomial translation from APW into AWW.

We finish with a discussion of how these techniques generalise to restricted classes of tree automata, leading to a partial collapse of the parity index-hierarchy.

We begin this section by fixing notations and definitions for ω -word and tree automata with parity acceptance conditions.

5.1. Automata on Words, Trees and Games.

Words, trees, games and graphs. A *word* over Σ is a (possibly infinite) sequence $w = w_0 \cdot w_1 \cdots$ of letters in Σ . We write $\text{suffixes}(w)$ for the set of suffixes of w (which includes w itself). We consider a *tree* to be an unranked infinite rooted tree in the graph-theoretic sense. A Σ -*tree* t is a tree together with a mapping of each of its nodes to a letter in Σ . We write $\text{subtrees}(t)$ for the set of Σ -subtrees of t (which includes t itself). Regular trees can be represented finitely as finite pointed Σ -labelled graphs (i.e., Kripke structures).

We identify the special *game alphabet* $\Gamma^d = \{A_i, E_i | i \in [0..d]\}$ that we use to represent parity games with priorities up to d . A Γ^d -labelled tree or graph is interpreted as the infinite or finite parity game on the arena consisting of the tree or graph where positions labelled E_i belong to Eve, positions labelled A_i belong to Adam, and positions labelled E_i or A_i have priority i .

Automata. Several different definitions of alternating tree automata exist. One variable is the branching type of the input trees: fixed or arbitrary, with or without an order on the successors. In the context of parity games, it is most natural to operate in a setting with arbitrary branching without an order on the successor nodes – that is, an automaton can not distinguish between the left and right branch of a binary tree, unless this is encoded in the labelling. Another difference in automata definitions is how flexible the transition condition is with respect to ϵ -transitions and the combination of path quantifiers (\Diamond, \Box) and boolean connectives (\vee, \wedge). Usually, all of these definitions give the same expressiveness ([Wil99, Proposition 1] and [Kir02, Remark 9.4]), except for the case of very restricted automata, in which they do not [BS18]. In our definition of alternating automata, there are no epsilon transitions and path quantifiers are applied directly on states.

An *alternating parity tree automaton* is a tuple $\langle \Sigma, Q, \iota, \delta, \Omega \rangle$ where Σ is a finite alphabet, Q is a finite set of states, $\iota \in Q$ is the initial state, $\delta : Q \times \Sigma \rightarrow \mathbf{B}^+(\{\Diamond, \Box\} \times Q)$ is the transition function, and $\Omega : Q \rightarrow I$ is a *priority assignment*.

Intuitively, given a state $q \in Q$ and a letter $\sigma \in \Sigma$, the transition function returns a positive boolean formula that defines which states the automaton should transition to, and whether to consider the next state at one non-deterministically chosen child (\Diamond), or at all of the children (\Box). Positive boolean formulas over $\{\Diamond, \Box\} \times Q$ are called *transition conditions*.

Formally, a *run* of an automaton with states Q over a Σ -tree t is a $(Q \times \Sigma)$ -tree r that assigns states to nodes of t along the transition function of \mathcal{A} . That is, there is a binary relation ρ that relates nodes of t and nodes of r and satisfies the following constraints.

- For every pair $(n, n') \in \rho$, n is a node of t , n' is a node of r , and if n is labeled σ then n' is labeled (\cdot, σ) . (The \cdot stands for an arbitrary value.) For every node n' of r , there is exactly one node n of t , such that $(n, n') \in \rho$.
- The roots of t and of r appear in exactly one pair, together, and r 's root is labeled (ι, \cdot) .
- For a node n of t with parent p , and a node n' of r with parent p' , if $(n, n') \in \rho$ then $(p, p') \in \rho$.
- Consider a node n of t , and a node n' of r labeled (q, σ) , such that $(n, n') \in \rho$. Let $\Phi = \delta(q, \sigma)$ be the transition condition of the state q over the letter σ . Then Φ should be satisfied by ρ as inductively defined below.
 - If $\Phi = \Diamond h$ then there exists a child c of n and a child c' of n' , such that $(c, c') \in \rho$ and c' is labeled (h, \cdot) .
 - If $\Phi = \Box h$ then for every child c of n , there is a child c' of n' , such that $(c, c') \in \rho$ and c' is labeled (h, \cdot) .
 - If $\Phi = b_1 \vee b_2$ (resp. $\Phi = b_1 \wedge b_2$), for transition conditions b_1 and b_2 , then b_1 or b_2 (resp. b_1 and b_2) should be satisfied.

$\Omega : Q \rightarrow I$ is a *priority function* that assigns to each state a priority from a set $I = [0..i]$ or $I = [1..i]$, for some $i \in \mathbb{N}$. A path is accepting if the maximal priority seen infinitely often in it is even.

A run is *accepting* if each of its paths satisfies the parity condition, namely, the highest priority seen infinitely often is even. An automaton \mathcal{A} *accepts* a tree if it has an accepting run on it; the language that it *recognises*, denoted by $L(\mathcal{A})$, is the set of trees that it accepts. A tree automaton accepts or rejects a graph according to whether it accepts or rejects its infinite tree unfolding. Two automata that recognise the same language are *equivalent*.

Büchi and *co-Büchi* automata are special cases of parity automata in which $I = \{1, 2\}$ and $I = \{0, 1\}$ respectively. An automaton is *weak* if every strongly connected component in the transition graph consists of states with either only odd priorities or only even priorities. Observe that a weak automaton can be seen as either a Büchi or co-Büchi automaton by using priorities 1 and 2 or 1 and 0 respectively.

The *size* of an automaton is the maximum of the alphabet length, the number of states, the number of subformulas in the transition function, and the acceptance condition's *index*, that is $|I|$. Observe that in alternating automata, the difference between the size of an automaton and the number of states in it can stem from a transition function that has exponentially many subformulas. (In stronger acceptance conditions, not considered here, the index might also be exponential in the number of states.)

How exactly nondeterminism in tree automata is defined also varies in the literature. In general, it only concerns the boolean connectives of the transition condition and not the path quantifiers (or directions, in ranked trees). We consider an alternating automaton to be *nondeterministic* (resp. *universal*) if its transition conditions only use the \vee (resp. \wedge) connective, in addition to the path quantifiers \Diamond and \Box .

A *word automaton* is simply a tree automaton that operates on unary trees, that is, ω -words. For word automata, the path quantifiers \Diamond and \Box are equivalent as there is always exactly one successor.

We will also mention *deterministic* word automata, in which the transition condition has no boolean connectives, *finite* automata, in which the only cycles are a rejecting sink

and an accepting sink, and *safety* automata, in which the only rejecting state is a rejecting sink.

The *class* of an automaton characterises its transition mode (deterministic, nondeterministic, or alternating), its acceptance condition, and whether it runs on words or trees. We often abbreviate automata classes by acronyms in $\{D, N, A\} \times \{W, B, C, P\} \times \{W, T\}$. The first letter stands for the transition mode; the second for the acceptance-condition (weak, Büchi, co-Büchi, and parity); and the third indicates whether the automaton runs on Words or on Trees. For example, AWW stands for an alternating weak automaton on words.

It is known that AWWs recognise all ω -regular word languages [Lin88], while AWTs do *not* recognise all regular tree languages, as they have the same expressiveness as alternation-free μ -calculus (AFMC) [MS95].

Definition 5.1 (Model-checking game). Given a Σ -tree t and an APT $\mathcal{A} = \langle \Sigma, Q, \iota, \delta, \Omega \rangle$, the model-checking game $\mathcal{G}(t, \mathcal{A})$ is the following parity game:

- Positions are $\text{subtrees}(t) \times (Q \cup B^+(\{\Diamond, \Box\} \times Q))$.
- For $a \in \Sigma$, a Σ -tree u whose root is labeled a , transition conditions b and b' , and state $q \in Q$, there is an edge from:
 - (u, q) to $(u, \delta(q, a))$
 - $(u, b \vee b')$ to (u, b) and (u, b')
 - $(u, b \wedge b')$ to (u, b) and (u, b')
 - $(u, \Diamond q)$ to (u', q) , for every child u' of u .
 - $(u, \Box q)$ to (u', q) , for every child u' of u .
- Positions $(\Box q, u)$ and $(b \wedge b', u)$ belong to Adam; other positions belong to Eve.
- A position (u, b) is of priority $\Omega(b)$ if b is a state q , and 0 otherwise.

Observe that the model-checking games of regular, i.e., finitely representable trees are finite since these trees only have finitely many distinct subtrees. Similarly, for a Kripke structure S , we write $\mathcal{G}(S, \mathcal{A})$ for the model-checking game $\mathcal{G}(t, \mathcal{A})$ where t is the tree-unfolding of S , and observe that it is finite.

Proposition 5.2. *An APT \mathcal{A} with initial state ι accepts a tree t if and only if Eve has a winning strategy in the model-checking game $\mathcal{G}(t, \mathcal{A})$ from (t, ι) .*

As a word is a special case of a tree, Definition 5.1, Proposition 5.2, and other results that will be shown on model-checking games apply also to word automata. In the word setting, the presentation of the model-checking game can be simplified: $\Diamond q$ and $\Box q$ are equivalent, and may thus be written q , and $Q \cup B^+(Q)$ is equal to $B^+(Q)$.

5.2. Automata for Register Games. We define for every APT \mathcal{A} and positive integer k , the parameterised version \mathcal{A}_k , which is an APT that will be shown to accept a tree t if and only if Eve wins the k -register game on $\mathcal{G}(t, \mathcal{A})$ starting from (t, ι) . The idea is to emulate the k -register game by keeping track of register configurations with a tuple $\bar{r} \in I^{k+1}$ that is updated according to which priorities are seen and Eve's register choices, which are represented as nondeterministic choices in \mathcal{A}_k . The outputs are captured by the priorities of the states of \mathcal{A}_k . Here we note a slight subtlety: In the k -register game on $\mathcal{G}(t, \mathcal{A})$, Eve chooses registers not only at positions (u, q) where q is a state of \mathcal{A} , but also at positions (u, b) where b is a boolean formula. In \mathcal{A}_k only states can have priorities (i.e., there can only be one priority per move in t) so we aggregate the outputs from these choices between

two states by taking the largest output into the priority of the next state—this is the third element $p \in [0..2k+1]$ of the states of \mathcal{A}_k .

Definition 5.3. Given an APT $\mathcal{A} = \langle \Sigma, Q, \iota, \delta, \Omega \rangle$ with $\Omega : Q \rightarrow I$ and a positive integer k , we define an APT $\mathcal{A}_k = \langle \Sigma, Q', \iota', \delta', \Omega' \rangle$ as follows:

- $Q' = Q \times I^{k+1} \times [0..2k+1]$
- $\iota' = (\iota, (0, \dots, 0), 0)$
- Ω' : For every $q \in Q, \bar{r} \in I^{k+1}$, and $p \in [0..2k+1]$, we have $\Omega'(q, \bar{r}, p) = p$.
- δ' : For every $q \in Q, \bar{r} \in I^{k+1}, p \in [0..2k+1]$, and $a \in \Sigma$, we have $\delta'((q, \bar{r}, p), a) = \bigvee_{i \in [0..k]} \text{move}(\delta(q, a), \text{new}_i(\bar{r}, \Omega(q)), \max(r_i, \Omega(q)))$ where for transition conditions b, b' :
 - $\text{new}_i(\bar{r}, p) = \bar{r}'$ where $r'_j = \max(r_j, p)$ for $j > i$, $r_i = p$ and $r_j = 0$ for $j < i$;
 - $\text{move}(\Diamond q', \bar{r}, p) = \bigvee_{i \in [0..k]} \Diamond(q', \text{new}_i(\bar{r}, 0), m)$ where $m = 2i$ if $\max(r_i, p)$ is even; $2i+1$ otherwise;
 - $\text{move}(\Box q', \bar{r}, p) = \bigvee_{i \in [0..k]} \Box(q', \text{new}_i(\bar{r}, 0), m)$ where $m = 2i$ if $\max(r_i, p)$ is even; $2i+1$ otherwise;
 - $\text{move}(b \wedge b', \bar{r}, p) = \bigvee_{i \in [0..k]} \text{move}(b, \text{new}_i(\bar{r}, 0), \max(r_i, p)) \wedge \text{move}(b', \text{new}_i(\bar{r}, 0), \max(r_i, p))$
 - $\text{move}(b \vee b', \bar{r}, p) = \bigvee_{i \in [0..k]} \text{move}(b, \text{new}_i(\bar{r}, 0), \max(r_i, p)) \vee \text{move}(b', \text{new}_i(\bar{r}, 0), \max(r_i, p))$

Lemma 5.4. *Given an APT \mathcal{A} and a positive integer k , the parameterised APT \mathcal{A}_k accepts a tree t if and only if Eve wins the k -register game on $\mathcal{G}(t, \mathcal{A})$ from (t, ι) .*

Proof. Recall that by Proposition 5.2, \mathcal{A}_k accepts a tree t if and only if Eve wins the parity game $\mathcal{G}(t, \mathcal{A}_k)$ from the initial position, which consists of \mathcal{A}_k 's initial state and t 's root. Thus, we should show that Eve wins $\mathcal{G}(t, \mathcal{A}_k)$ from the initial position if and only if she wins the k -register game on $\mathcal{G}(t, \mathcal{A})$ from the initial position. The intuition for the equivalence between the games is that $\mathcal{G}(t, \mathcal{A}_k)$ encodes the k -register game on $\mathcal{G}(t, \mathcal{A})$ by having the register-configuration in the state space, Eve's register choices as new disjunctions, and the highest output between two states as priorities.

Positions of $\mathcal{G}(t, \mathcal{A}_k)$ are in

$$\text{subtrees}(t) \times \left((Q \times I^k \times [1..2k+1]) \cup B^+(\{\Diamond, \Box\} \times Q \times I^k \times [1..2k+1]) \right)$$

while positions of the k -register game on $\mathcal{G}(t, \mathcal{A})$ are in $\text{subtrees}(t) \times \left(Q \cup B^+(\{\Diamond, \Box\} \times Q) \right) \times I^k$.

$\mathcal{G}(t, \mathcal{A}_k)$ begins at $(t, (\iota, (0, \dots, 0), 0))$ while the k -register game on $\mathcal{G}(t, \mathcal{A})$ begins at $(t, \iota, (0, \dots, 0))$.

Now, observe that in the k -register game on $\mathcal{G}(t, \mathcal{A})$ at position (u, q, \bar{r}) , for a state $q \in Q$ and a Σ -tree u whose root is labeled a , Eve has to choose a register $i \in [0..k]$ before the parity game proceeds to $(u, \delta(q, a))$ with register configuration \bar{r}' , which is exactly $\text{new}_i(\bar{r}, \Omega(q))$. Similarly in $\mathcal{G}(t, \mathcal{A}_k)$, from (u, q, \bar{x}, p) Eve chooses $i \in [0..k]$ before proceeding with the transition condition $\delta(q, a)$ with register configuration $\text{new}_i(\bar{r}, \Omega(q))$. In the register game on $\mathcal{G}(t, \mathcal{A})$, this produces an output $2i$ if $\max(r_i, \Omega(q))$ is even; $2i+1$ otherwise. In $\mathcal{G}(t, \mathcal{A}_k)$ the value $\max(r_i, \Omega(q))$ is kept in the state of the successor position.

When the games proceed, in both the register game on $\mathcal{G}(t, \mathcal{A})$ and in the parity game on $\mathcal{G}(t, \mathcal{A}_k)$, the decision falls to Adam if $\delta(q, a)$ is a conjunction or prefixed by \Box and to Eve otherwise. Then, between each subformula of the transition condition, Eve again has a choice of register in both games. In the register game on $\mathcal{G}(t, \mathcal{A})$, this produces an output $2i$ if $\max(r_i, 0)$ (that is r_i) is even; $2i+1$ otherwise. The largest output between two positions

in $Q \times \text{subtrees}(t)$ of $\mathcal{G}(t, \mathcal{A})$ is recorded via the third element of the automata-component of $\mathcal{G}(t, \mathcal{A}_k)$, which determines the priority of the successor state.

Then a winning strategy for Eve in one game translates into a winning strategy for Eve in the other game by mapping Eve's choices of registers in the register game on $\mathcal{G}(t, \mathcal{A})$ to her choices in $\mathcal{G}(t, \mathcal{A}_k)$ at disjunctions over $[0..k]$ in the transition condition of \mathcal{A} and vice versa, and mapping her choices in the underlying parity game $\mathcal{G}(t, \mathcal{A})$ to the remaining choices in $\mathcal{G}(t, \mathcal{A}_k)$, and vice-versa. \square

Remark 5.5. The modal μ -minded reader can get an idea of the equivalent modal μ formula from [Leh18] where the definability of register-games was presented from a logic-perspective, rather than in terms of automata. The following definition in particular is the automata-equivalent of the canonical modal μ formulas that witness the strictness of the alternation hierarchy [Bra98].

Definition 5.6. For finite positive d , we identify \mathcal{P}^d , the alternating parity automaton on Γ^d -trees that recognises parity games with priorities up to d in which Eve has a winning strategy. This is simply an automaton with states $0, 1 \dots d$ where the priority of a state i is i and the transition condition is $\delta(q, E_i) = \Diamond i$ and $\delta(q, A_i) = \Box i$.

By combining Theorem 4.2 and Lemma 5.4, we can state the quasi-polynomial complexity of parity games in terms of *descriptive complexity*.

Lemma 5.7. *There is a finite automaton of size $n^{O((\log n)^2)}$ that recognises the language of pointed parity games of size up to n in which Eve has a winning strategy.*

Proof. \mathcal{P}^d with d states and d priorities recognises the language of parity games with an initial state from which Eve has a winning strategy; more precisely, the model-checking game $\mathcal{G}(t, \mathcal{P}^d)$ for any Γ^d -tree t is identical to t once we collapse positions with a unique successor. It follows that t and $\mathcal{G}(t, \mathcal{P}^d)$ have identical register-index. Then, from Theorem 4.2 and Lemma 5.4 $\mathcal{P}_{\log n+1}^n$ recognises the same language.

On parity games of bounded size, a finite automaton suffices: one can further reduce the number of priorities of the automaton by allowing each odd (even) priority to be visited only n times without a higher priority occurring before transitioning to the rejecting (accepting) sink. Since the number of priorities in $\mathcal{P}_{\log n+1}^n$ is logarithmic in n , this only adds a factor of $n^{\log n}$ to the state-space of the automaton. \square

The automata-definability of the register-game also implies that the register-index is a *bisimulation invariant* measure of complexity for parity games.

Lemma 5.8. *Register-index is bisimulation-invariant.*

Proof. \mathcal{P}_k^d accepts a parity game G with priorities up to d whenever Eve wins $\mathcal{R}_E^k(G)$. Dually, we can define $\text{co}\mathcal{P}_k^d$ that accepts parity games G with priorities up to d whenever Adam wins $\mathcal{R}_A^G(k)$ (note that this is not the complement of \mathcal{P}_k^d). Recall that a parity game has register-index at most k if $\mathcal{R}_A^G(k)$ and $\mathcal{R}_E^G(k)$ have the same winner.

Then, we can build an automaton that accepts exactly parity games with priorities up to d that have register-index at most k : $(\neg \text{co}\mathcal{P}_k^d \cap \mathcal{P}_k^d) \cup (\text{co}\mathcal{P}_k^d \cap \neg \mathcal{P}_k^d)$. Since alternating parity automata, like formulas of the modal μ -calculus, are bisimulation invariant, so is register-index. \square

5.3. Register Games and Word Automata. We turn our attention to automata over ω -words. On words, deterministic parity automata, non-deterministic Büchi automata and alternating weak automata all are expressive enough to capture all ω -regular languages. While the cost of determinisation of alternating automata is known to be double-exponential, our understanding of the trade-offs in conciseness between different acceptance conditions is incomplete. For instance, ABW are at most quadratically more concise than equivalent AWW, but the current corresponding lower bound is at $\Omega(n \log n)$. Until recently the best translation from APW into AWW was exponential; here we improve it to quasi-polynomial; the lower bound remains at $\Omega(n \log n)$.

In this section we use the previously defined register-automata to define a quasi-polynomial translation from APW to AWW. It is based on observing that the model-checking parity games of an automaton on a regular word, which can be described by lasso Kripke structure, have a register-index that depends solely on the automaton and is at most logarithmic in its size. Given that equivalence over regular words implies equivalence over all words, this will suffice to turn APW into equivalent APW with a logarithmic number of priorities and of quasi-polynomial size; from there, Kupferman and Vardi's classic translation into weak [KV98] produces a weak automaton which is also of only quasi-polynomial size.

We begin by showing that the register-index of $\mathcal{G}(w, \mathcal{A})$, for an ultimately periodic word w , is independent of w and logarithmic in \mathcal{A} . This is a consequence of Theorem 4.2 and the fact that w is represented by a Kripke structure with a single cycle.

Lemma 5.9. *Given an ultimately periodic word $w = uc^\omega$ and an APW \mathcal{A} with n states, the parity game $\mathcal{G}(w, \mathcal{A})$ has register-index at most $1 + \log n$.*

Proof. We show that the dc-size of $\mathcal{G}(w, \mathcal{A})$ is at most n . Then, from Theorem 4.2, its register-index is at most $1 + \log n$.

First note that all cycles of $\mathcal{G}(w, \mathcal{A})$ occur in the subgame $\mathcal{G}(c^\omega, \mathcal{A})$. We consider the graph H that is derived from $\mathcal{G}(c^\omega, \mathcal{A})$ by ignoring the intermediate positions (v, b) , where b is a boolean formula between positions of the form (v, q) , for a state q . That is, let H be the graph consisting of just the vertices (v, q) of $\mathcal{G}(c^\omega, \mathcal{A})$, where q is a state. The edges of H connect positions (v, q) and (v', q') if (v', q') is reachable from (v, q) in $\mathcal{G}(c^\omega, \mathcal{A})$ directly, that is, with a path which does not visit yet another position (v'', q'') where q'' is a state.

$\mathcal{G}(c^\omega, \mathcal{A})$ has dc-size no larger than the graph H : a set of disjoint cycles $\mathcal{G}(c^\omega, \mathcal{A})$ induces a set of disjoint cycles in H . If m is the size of the cycle c , then each cycle of H must be of size at least m . H is of size at most mn , so the dc-size of H , and therefore of $\mathcal{G}(w, \mathcal{A})$, is at most n . \square

Then \mathcal{A} is equivalent to its $1 + \log n$ parameterised version; the main result follows by applying the existing transformation – exponential in the number of priorities – to $\mathcal{A}_{1+\log n}$.

Lemma 5.10. *Every APW A is equivalent to its parameterised version A_k , for $k = 1 + \log n$.*

Proof. Since two ω -regular languages are equivalent if they agree on the set of ultimately periodic words [McN66], it suffices to argue that A and A_k agree on ultimately periodic words.

From Lemma 5.4, A_k accepts an ultimately periodic word w if and only if Eve wins the k -register game on $\mathcal{G}(w, \mathcal{A})$. From Lemma 5.9, this is the case exactly when Eve wins the parity game on $\mathcal{G}(w, \mathcal{A})$, that is, when A accepts w . \square

Theorem 5.11. *The size blow-up and state blow-up involved in translating alternating parity word automata to alternating weak word automata is at most quasi-polynomial. In particular, every APW \mathcal{A} of size (resp. number of states) n is equivalent to an AWW of size (resp. number of states) $2^{O((\log n)^3)}$.*

Proof. From Lemma 5.10, an APW \mathcal{A} with n states and d priorities is equivalent to its parameterised APW \mathcal{A}_k for $k = 1 + \log n$, having $n \cdot d^k \cdot (2k + 1)$ states and $2k + 1$ priorities. \mathcal{A}_k can then be turned into a weak automaton using standard techniques [KV98] with a $O(m^{d'})$ blow-up, where m is the number of states and d' the number of priorities, which yields an AWW with $2^{O((\log n)^3)}$ many states, since m is here in $O(kn^{k+1}) \leq 2^{O((\log n)^2)}$ and d' is $2k + 1 \in O(\log n)$.

In case that the size of \mathcal{A} is dominated by the size e of its transition function, namely when $e > n$, observe that the parameter k , the number of states in \mathcal{A}_k , and the number of priorities in \mathcal{A}_k do not depend on e , while the size of \mathcal{A}_k 's transition function is in $O(k^2 e d^k) \leq 2^{O((\log e)^2)}$. Since the translation in [KV98] does not blow up the transition-function size more than it blows up the number of states, we end up with an AWW of size in $2^{O((\log e)^3)}$. \square

5.4. Register Games and Tree automata. The results shown above for word automata do not extend to tree automata. First of all, while the parity hierarchy collapses in alternating word automata, it is strict in alternating tree automata [Lin88, Bra98, BL18]: For every positive integer n , there is an APT \mathcal{A} with $O(n)$ states and $O(n)$ priorities, such that there is no APT equivalent to \mathcal{A} with less than n priorities. Hence the weak condition does not suffice to capture all tree-languages captures by parity automata, so a translation from APT to AWT does not exist for all APT. However, even for APT that are equivalent to an AWT, such a translation is at least exponential [BL18], in contradistinction to the quasipolynomial translation of APW to AWW.

The method used for word automata fails for trees because in the word setting the register-index of the model-checking game of an (APW) automaton and an input (word) only depends on the automaton, while in the tree setting it depends on both the (APT) automaton and the input (tree).

For our analysis, we focus on regular trees and words. Indeed, it is known that two ω -regular word or tree automata are equivalent if and only if they are equivalent with respect to words or trees generated by finite Kripke structures: since alternating automata are closed under negation and intersection this is, for example, a consequence of the finite model theorem that states that all μ -formulas that are satisfiable, and therefore all non-empty APTs, are satisfied by (accept) a finite structure [Koz88].

Regular trees are the unfoldings of arbitrary Kripke structures while ultimately periodic words are unfoldings of the very restrictive ‘lasso’ Kripke structures, namely Kripke structures in which all states have out-degree 1. A natural question is then to consider the index hierarchy of alternating automata on entities that are “between” words and trees—that is, infinite trees generate by Kripke structures that are more complex than lassos, but more restrictive than trees. Examples of such classes are “flat Kripke structures” [DDS12], in which every maximal strongly connected component (MSCC) has a single cycle, “weak Kripke structures” [KF11], in which MSCCs cannot have two vertex-disjoint cycles and Kripke structures with finite Cantor–Bendixson rank [BW18].

We show that the parity hierarchy *collapses logarithmically* for alternating automata on a class of Kripke structures that subsumes in particular Kripke structures that are flat, weak or of finite Cantor–Bendixson rank. That is, for every APT \mathcal{A} with n states, there is an APT \mathcal{A}' with $O(\log n)$ priorities, such that \mathcal{A} and \mathcal{A}' are equivalent with respect to trees generated by Kripke structures of this class. The class consists of Kripke structures in which every MSCC has a bounded number of disjoint cycles.

To show this result, we first recall the “minimum feedback vertex set” measure of graphs, and prove that the register index of the model-checking game of an APT \mathcal{A} and a Kripke structure S depends on the number of states in \mathcal{A} and this measure in S .

Definition 5.12. For a directed graph G , let *fvs-size* be the size of a minimum feedback vertex set of G , where a feedback vertex set contains at least one vertex of every cycle in G .

We now show that the dc-size of $\mathcal{G}(t, \mathcal{A})$, for a tree t that represents the computations of a finite Kripke structure with fvs-size d in each of its MSCCs, is logarithmic in the number of states in \mathcal{A} times d . That is, it is not the number of states in the Kripke structure that influences the register-index of the model-checking game, but only its minimum feedback vertex set.

Lemma 5.13. *Given a Kripke structure S with fvs-size up to d in each of its MSCCs and an APT \mathcal{A} with n states, the parity game $\mathcal{G}(S, \mathcal{A})$ has register-index at most $1 + \log dn$.*

Proof. We show that the dc-size of $\mathcal{G}(S, \mathcal{A})$ is at most dn . Then, by Lemma 4.2, its register-index is at most $1 + \log dn$.

First note that every MSCC of $\mathcal{G}(S, \mathcal{A})$ results from a single MSCC of S and a single MSCC of \mathcal{A} . It is therefore enough to assume that both S and \mathcal{A} consist of a single MSCC.

We consider the graph H that is derived from $\mathcal{G}(S, \mathcal{A})$ by ignoring the intermediate positions (v, b) , where b is a boolean formula between positions of the form (v, q) , for a state q . That is, let H be the graph consisting of just the vertices (v, q) of $\mathcal{G}(S, \mathcal{A})$, where q is a state. The edges of H connect positions (v, q) and (v', q') if (v', q') is reachable from (v, q) in $\mathcal{G}(S, \mathcal{A})$ directly, that is, with a path which does not visit yet another position (v'', q'') where q'' is a state.

Observe that $\mathcal{G}(S, \mathcal{A})$ has dc-size no larger than the graph H : a set of disjoint cycles in $\mathcal{G}(S, \mathcal{A})$ induces a set of disjoint cycles in H .

Let F be a feedback vertex set of S , having up to d vertices. Since every cycle C in H corresponds to some cycle of S , it must contain a vertex (v, q) , such that q is a state of \mathcal{A} and $v \in F$. Hence, for every $v \in F$, there are up to n vertex-disjoint cycles in H that have a vertex of the form (v, q) . Therefore, there are up to $|F|n \leq dn$ disjoint cycles in H . □

Then we get the logarithmic collapse of the parity hierarchy with respect to Kripke structures with bounded fvs-size.

Theorem 5.14. *Every APT \mathcal{A} with n states is equivalent to an APT \mathcal{A}' with $1 + \log dn$ priorities with respect to Kripke structures with fvs-size of up to d in each of their MSCCs.*

Proof. Let $k = 1 + \log dn$. Setting \mathcal{A}' to be the k -parameterised version \mathcal{A}_k of \mathcal{A} fits the bill. Indeed: By Proposition 5.2, \mathcal{A} accepts a Kripke structure S iff Eve wins $\mathcal{G}(S, \mathcal{A})$. By Lemma 5.13, for every Kripke structure S with fvs-size of up to d in each of its MSCCs, the register index of $\mathcal{G}(S, \mathcal{A})$ is at most k . Hence, Eve wins $\mathcal{G}(S, \mathcal{A})$ iff she wins the k -register game on $\mathcal{G}(S, \mathcal{A})$, and by Lemma 5.4, this happens iff \mathcal{A}_k accepts S . □

There are several other graph measures that are known to be bounded when the fvs-size is bounded, such as the number of disjoint cycles (with respect to either vertices or edges) [EP65]. Accordingly, the parity hierarchy logarithmically-collapses with respect to Kripke structures whose MSCCs have a bounded number of disjoint cycles.

Corollary 5.15. *Every APT with n states is equivalent to an APT with $O(\log n)$ priorities with respect to Kripke structures whose MSCCs are bounded in one of the following measures: minimum feedback vertex set, minimum feedback edge set, maximal number of vertex-disjoint cycles, and maximal number of edge-disjoint cycles.*

Proof. Observe that the minimum feedback vertex set is always smaller than or equal to the minimum feedback edge set, and likewise the maximal number of vertex-disjoint cycles is smaller than or equal to the maximal number of edge-disjoint cycles. Hence, it is enough to consider the vertex-version of these measures.

The result with respect to the minimum feedback vertex set follows directly from Theorem 5.14. The result with respect to the maximal number of disjoint cycles follows from Theorem 5.14 and the Erdős-Pósa Theorem [EP65], which states that the maximal number of disjoint cycles is logarithmically bounded in the minimum feedback vertex set. \square

It may well be the case that this collapse of the index-hierarchy on these structures goes further, all the way to weak, as on words, using similar techniques to Kupferman and Vardi [KV98]. We leave this as an open problem, as well as the related question of what is the simplest class of trees on which the hierarchy is strict.

6. OTHER QUASI-POLYNOMIAL AUTOMATA

We have already seen two different automata-theoretic constructions that generalise solving parity games: tree automata that recognise parity games of bounded size in which Eve has a winning strategy, and translations from alternating parity word automata to alternating weak automata. Bojańczyk and Czerwiński [BC18] have proposed a third one, based on *deterministic or good-for-games safety word automata*, which captures both Calude et al.’s and Jurdziński and Lazić’s parity game algorithms.

The aim of this section is to clarify how these three generalisations relate to each other, and how existing quasi-polynomial algorithms fit into the picture. In particular, existing comparisons between Calude et al.’s, Jurdziński and Lazić’s and the register-game algorithm consider them from the perspective of word automata, in particular *safety* word automata. Here we complement this discussion with a tree-automata perspective and consider the question of when an automata-theoretic solution to parity games also generalises to word-automata translations, in the spirit of Section 5.3. We argue that considering parity games with a bounded number of disjoint cycles, rather than bounded size, is key to this type of generalisation. We conclude this discussion with a brief note on good-for-gameness.

6.1. Descriptive solutions to parity games. The register-index approach is the first quasi-polynomial algorithm to be explicitly presented in term of tree automata and, equivalently, the μ calculus. However, Bojańczyk and Czerwiński [BC18] have given an alternative presentation of both Calude et al.’s and Jurdziński and Lazić’s quasi-polynomial algorithms in terms of *deterministic safety word automata*. Deterministic safety automata are not expressive enough to capture the parity condition; Instead, Bojańczyk and Czerwiński consider automata that separate plays that agree with deterministic winning strategies for each

player in parity games of size up to n with up to d priorities. Finding a deterministic safety automaton of size $f(n, d)$ that separates these word languages suffices to solve parity games in time polynomial in $f(n, d)$. If the separation condition is strengthened to a separation between the language of plays that agree with a positional winning strategy for Eve in some parity game of size n with d priorities and the language of plays that do not satisfy the parity condition, then a quasi-polynomial lower bound applies [CDF⁺19].

We now phrase the idea that a separating word automaton suffices to solve parity games in terms of tree automata. Indeed, from Bojańczyk and Czerwiński’s insight, it is only a small step to concluding that from a deterministic (or even any *good for games*) separating word automaton, one can build a tree automaton that recognises parity game arenas of size up to n in which Eve has a winning strategy.

Proposition 6.1. *Let \mathcal{A} be a deterministic parity word automaton over the alphabet $[0..d]$ that:*

- *Accepts words that agree with a positional winning strategy for Eve in some parity game of size up to n with up to d priorities, and*
- *Rejects words that agree with a positional winning strategy for Adam in some parity game of size up to n with up to d priorities.*

Then, there is an alternating tree automaton \mathcal{A}' of the same size and acceptance condition as \mathcal{A} which recognises parity games winning for Eve of size up to n with up to p priorities.

Proof. While \mathcal{A} operates on an alphabet of priorities, \mathcal{A}' operates on the richer game alphabet Γ^d that also encodes the ownership of nodes. \mathcal{A}' has the same state-space, initial state and priority assignment as \mathcal{A} . The only difference is the transition function of \mathcal{A}' which is simply $\delta'(q, E_i) = \Diamond\delta(q, i)$ and $\delta'(q, A_i) = \Box\delta(q, i)$. In other words, \mathcal{A}' gives the decision of which branch to choose to the player who owns the current position, but otherwise operates exactly as \mathcal{A} . Whichever player has a winning strategy in a parity game G of size n with up to d priorities encoded as a Γ -graph can copy their positional winning strategy in the model-checking game $\mathcal{G}(G, \mathcal{A}')$; since \mathcal{A} separates the plays resulting from such strategies, \mathcal{A}' will accept if and only if Eve has a winning strategy in G . \square

In this sense, these quasi-polynomial algorithms are also descriptive solutions to parity games. In the sequel, to better understand some of the differences between the algorithms, we discuss what more is needed for a descriptive solution to be able to be used to turn alternating parity word automata into weak automata.

6.2. From automata to automata transformations. Despite each of these three approaches to solving parity games in quasi-polynomial time being automata-definable in this way, a notable difference between (at least our current understanding thereof) is that the register-index approach, as seen in Section 5.3, is also suited for handling the parity games of unbounded size that stem from word automata, thus allowing for a quasi-polynomial translation from alternating parity word automata to alternating weak automata. The same generalisation is not immediate for Calude et al.’s and Jurdziński and Lazić’s algorithms; in particular, a safety-automata based approach is unlikely to suffice since safety automata are not as expressive as parity automata. This raises the following question: when does a descriptive solution to parity games, that is, an automaton that recognises the winning

regions of parity games of bounded size, imply a translation from alternating parity word automata into weak automata?

We propose infinite directed acyclic graphs (dags) of bounded width as a key ingredient. The model-checking games for word automata take this shape, which make them an interesting stepping stone between words and trees. Indeed, as we shall see, all tree automata that recognise infinite parity games of bounded width in which Eve has a winning strategy can be used to turn alternating parity word automata into word automata of the same acceptance condition. The translation is easy: it consists of building the synchronised composition of the two automata.

The technical details of the synchronised composition are cumbersome, but the idea is straight-forward: the synchronised composition of \mathcal{B} and \mathcal{A} is an automaton that accepts a tree t if and only if \mathcal{A} accepts the model-checking game of \mathcal{B} and t when viewed as a Γ -tree.

Definition 6.2 (Synchronised Composition). Let $\mathcal{A} = (\Sigma^A, Q^A, \iota^A, \delta^A, \Omega^A)$ be an APT with priorities up to d . Let $\mathcal{B} = (\Gamma^d, Q^B, \iota^B, \delta^B, \Omega^B)$ be an APT over the game alphabet Γ^d .

The synchronised product $\mathcal{B} \times \mathcal{A}$ is defined as:

- State space $Q^A \times Q^B$;
- Alphabet Σ^A ;
- $\Omega(q_A, q_B) = \Omega^B(q_B)$.
- Transition relation: $\delta((q_A, q_B), \alpha) = f^\alpha(q_A, \delta^B(q_B, \text{label}(q_A)))$ where:
 - $f^\alpha(b, p) = f^\alpha(b, \delta^B(p, \text{label}(b)))$ where $p \in Q^B$
 - $f^\alpha(q_A, \Diamond p) = \Diamond f^\alpha(\delta^A(q_A, \alpha), p)$
 - $f^\alpha(q_A, \Box p) = \Box f^\alpha(\delta^A(q_A, \alpha), p)$
 - $f^\alpha(b \wedge b', \Diamond p) = f^\alpha(b, p) \vee f^\alpha(b', p)$
 - $f^\alpha(b \vee b', \Diamond p) = f^\alpha(b, p) \vee f^\alpha(b', p)$
 - $f^\alpha(\Diamond q, \Diamond p) = \Diamond(q, p)$
 - $f^\alpha(\Box q, \Diamond p) = \Diamond(q, p)$
 - $f^\alpha(b \wedge b', \Box p) = f^\alpha(b, p) \wedge f^\alpha(b', p)$
 - $f^\alpha(b \vee b', \Box p) = f^\alpha(b, p) \wedge f^\alpha(b', p)$
 - $f^\alpha(\Diamond q, \Box p) = \Box(q, p)$
 - $f^\alpha(\Box q, \Box p) = \Box(q, p)$
 - $f^\alpha(b, c \vee c') = f^\alpha(b, c) \vee f^\alpha(b, c')$
 - $f^\alpha(b, c \wedge c') = f^\alpha(b, c) \wedge f^\alpha(b, c')$
 - $\text{label}(b \wedge b') = \text{label}(\Box q) = A_0$
 - $\text{label}(b \vee b') = \text{label}(\Diamond q) = E_0$
 - $\text{label}(q) = E_{\Omega^A(q)}$

Lemma 6.3. $\mathcal{A} \times \mathcal{B}$ accepts a Σ^A -tree t if and only if \mathcal{B} accepts the model-checking game $\mathcal{G}(t, \mathcal{A})$ seen as a Γ^d -tree for d the maximal priority in \mathcal{A} .

Proof. We show that $\mathcal{G}(\mathcal{G}(t, \mathcal{A}), \mathcal{B})$ and $\mathcal{G}(t, \mathcal{A} \times \mathcal{B})$ have the same winner. The transition relation is designed so that $\mathcal{G}(\mathcal{G}(t, \mathcal{A}), \mathcal{B})$ is identical to $\mathcal{G}(t, \mathcal{A} \times \mathcal{B})$. More precisely, we identify position $((t, q_A), q_B)$ and $((t, b_A), b_B)$ in $\mathcal{G}(\mathcal{G}(t, \mathcal{A}), \mathcal{B})$ with $(t, (q_A, q_B))$ and $(t, f^\alpha(b_A, b_B))$ in $\mathcal{G}(t, \mathcal{A} \times \mathcal{B})$, respectively, where α is the label of t . We now show that this mapping preserves successors, position ownership and parity; the preservation of winner follows.

Preservation of successors: We observe:

- $((t, q_A), q_B)$ has successor $((t, q_A), \delta^B(q_B, E_{\Omega(q_A)}))$

- $(t, (q_A, q_B))$ has successor $(t, f^\alpha(q_A, \delta^B(q_B, E_{\Omega(q_A)})))$
- $((t, b \wedge b'), \Diamond p)$ has successors $((t, b), p)$ and $((t, b'), p)$
- $(t, f^\alpha(b \wedge b', \Diamond p))$ has successors $(t, f^\alpha(b, p))$ and $((t, f^\alpha(b', p)))$
- $((t, \Diamond q), \Diamond p)$ has successors $((t', q), p)$ for all children t' of t .
- $(t, f(\Diamond q, \Diamond p))$ has successors $(t', (q, p))$ for all children t' of t .
- Similarly for other combinations of modalities and boolean operators.

Preservation of ownership: Eve owns position $((t, b_a), b_B)$ in $\mathcal{G}(\mathcal{G}(t, A), B)$ whenever b_A is a disjunction or a \Diamond -formula, and positions with a unique successor. Similarly, Eve owns $(t, f^\alpha(b_a, b_B))$ in $\mathcal{G}(t, \mathcal{A} \times \mathcal{B})$ whenever b_B is a disjunction or \Diamond -formula, and positions with a unique successor.

Preservation of priorities: The priority of both $((t, q_A), q_B)$ and $(t, (q_A, q_B))$ is $\Omega^B(p)$ and the priority of other positions is 0.

Then, since $\mathcal{G}(\mathcal{G}(t, \mathcal{A}), \mathcal{B})$ and $\mathcal{G}(t, \mathcal{A} \times \mathcal{B})$ must have the same winner, $\mathcal{A} \times \mathcal{B}$ accepts a tree t if and only if \mathcal{B} accepts $\mathcal{G}(\mathcal{A}, \mathcal{B})$. \square

Lemma 6.4. $\mathcal{A} \times \mathcal{B}$ has the same acceptance condition as \mathcal{B} .

Proof. $\mathcal{A} \times \mathcal{B}$ clearly preserves the priorities of \mathcal{B} . Furthermore, it preserves the non-reachability of states: if q'_B is not reachable from q_B , then (q_A, q'_B) is not reachable from any (q'_A, q'_B) . Hence, if \mathcal{B} is weak, then $\mathcal{A} \times \mathcal{B}$ is weak. \square

Proposition 6.5. *If for a class C of trees, for all $t \in C$, \mathcal{A} accepts $G(t, \mathcal{B})$ if and only if \mathcal{B} accepts t , then $\mathcal{B} \times \mathcal{A}$ is equivalent to \mathcal{B} over C .*

Definition 6.6. An infinite dag is of bounded width m if its vertices can be partitioned into sets L_0, L_1, L_2, \dots no larger than n such that every edge goes from some layer L_i to the next layer L_{i+1} .

Corollary 6.7. *An alternating parity automaton $\mathcal{B}_{(m,d)}$ that over regular parity game dags of bounded width m with up to d priorities recognises those in which Eve has a winning strategy induces a translation from alternating parity word automata to alternating automata with the acceptance condition of $\mathcal{B}_{(m,d)}$ with blow-up linear in $|B_{(m,d)}|$.*

Proof. The model-checking game of a ultimately periodic word w and an APW \mathcal{A} with up to d priorities is a regular parity game dag of width $|\mathcal{A}|$ with up to d priorities; hence $\mathcal{B}_{(|\mathcal{A}|,d)}$ recognises the winning regions of the model-checking games of \mathcal{A} over ultimately periodic words. The automaton $\mathcal{A} \times \mathcal{B}_{(|\mathcal{A}|,d)}$ is, from Proposition 6.5, equivalent to \mathcal{A} on ultimately periodic words and equivalence over ultimately periodic words implies equivalence over all words. $\mathcal{A} \times \mathcal{B}_{(|\mathcal{A}|,d)}$ is therefore equivalent to \mathcal{A} and has the acceptance condition of $\mathcal{B}_{(|\mathcal{A}|,d)}$. The blow-up is linear in $\mathcal{B}_{(|\mathcal{A}|,d)}$. \square

We now observe that \mathcal{P}_k^d from Section 5 is indeed an automaton that recognises the winning regions of dags of width m when $k = 1 + \log m$ and d priorities. Recall that it recognises parity games in which Eve has a winning strategy over finite parity games with up to m disjoint cycles with d priorities. Furthermore, a regular parity game dag of width up to m can be represented by a finite graph with up to m disjoint cycles: indeed, if a graph unfolds into a dag of width m , its disjoint cycles unfold into infinite disjoint paths, of which there can be only m .

From these observations, it seems that reasoning about parity games of arbitrary size but with a bounded number of disjoint cycles, or infinite parity games of bounded width,

rather than only finite parity games, is a key distinction between algorithms for solving finite parity games and translations from one type of automata into another.

6.3. Good for games. So far, we have discussed bounded-width dags as an interesting intermediate form of object between words and trees; they allow us to use tree automata to design translations for word-automata. In some sense, good-for-games automata help us reason about the reverse direction—building tree automata out of word automata.

Intuitively, *good for games* word automata are automata \mathcal{A} over an alphabet Σ for which, given any Σ -labelled arena G —that is, a graph of which the positions are partitioned between two players and labelled with Σ —Eve wins the synchronised product of G and \mathcal{A} if and only if she has a strategy σ in G such that every path that agrees with σ forms a word accepted by \mathcal{A} . Deterministic automata are trivially good for games, while for non-deterministic automata the notion of *history determinism* implies and is sometimes used instead of good-for-gameness [HP06]. For alternating automata, the notion of good-for-gameness, although briefly considered in [Col13], has not been studied in depth.

Observe that good-for-games automata are exactly the word automata which can be turned into alternating tree automata in the way outlined in Proposition 6.1. Indeed, Proposition 6.1 hinges on the fact that it doesn’t matter whether the automaton reads the priorities *while* Adam and Eve negotiate the parity game—like in the synchronised product—or *after* both have chosen their strategies. This is trivially true for a deterministic automaton; the requirement of good-for-gameness allows this proposition to extend beyond deterministic automata.

The register automaton \mathcal{P}_k^d that recognises whether Eve wins the k -register game, unlike the deterministic automata presented by Bojańczyk and Czerwiński [BC18], is not good for games. First, observe that for $k > 0$, the automaton \mathcal{P}_k^d on words is trivially a separating automaton for words with up to d priorities: indeed, all words have register-index 1. This automaton is obviously not deterministic. It is not good for games either since there are parity games of register-index larger than k in which Eve has a winning strategy. It is only “good for small games”—or even “good for thin games”—in the sense that when interpreted on games, it only recognises parity games of size (or even width) up to 2^{k-1} in which Eve has a winning strategy.

7. CONCLUSIONS

Throughout this article we have presented an automata-theoretic take on solving parity games in quasi-polynomial time based on the notion of register games. This perspective enabled us to go beyond finite parity games, and study automata transformations and the index hierarchy.

Solving parity games in practice. This article has focused on the automata-theoretic aspect of register games, rather than how they can be used to solve parity games in practice. We would therefore forgive the reader for questioning the practicality of this quasi-polynomial algorithm, which, among its less flattering features, has quasi-polynomial space complexity.

However, there is much unexplored potential both for improving the practical viability of this algorithm, and for using the insights of this analysis to improve existing solvers.

Indeed, register-games have shown us that building parity games with truly complex winning strategies—in the sense of requiring more than a small constant number of register

in the register game—is subtle business: see the construction in Lemma 3.7. We therefore conjecture that the parameterised version of this algorithm, which solves parity games of register-index 1, is a plausible candidate for solving *most* reasonable parity games in polynomial time, and perhaps also in practice. Increasing the parameter to 2 or 3 could ensure it only fails for purpose-built counter-examples. Furthermore, the insight that even complex-looking parity games tend to have low register-index could be useful in itself, for optimising existing solvers, which are not necessarily quasi-polynomial, but still more effective in practice.

To address the space-complexity issue, we propose two potential solutions. On one hand, a similar approach to the one proposed by Fearnley et al. [FJS⁺17], which reduced the space-complexity of Calude et al.’s algorithm, seems likely to work for the register-index approach, too. Furthermore, the register-index approach seems suited for a symbolic implementation: given a symbolically represented parity game, the k -register game can also be represented symbolically without significant blow-up. Such an approach would bypass the space-complexity of the algorithm for parity games that benefit from concise symbolic representations.

Further open problems. We have, throughout this article, pointed to some problems that remain open. The most obvious, of course, is the complexity of solving parity games. The discussion of Section 6 suggests some possible directions, both for improving our current automata and lower bounds: the existing quasi-polynomial algorithms, when seen as automata, only use non-determinism; we currently don’t know whether automata that exploit alternations could yield more conciseness, both for the APT to AWW translation and for solving parity games.

We have also mentioned the parity index problem, which is intimately connected to the automata-theoretic concerns that this article touches upon. The question of when exactly, when moving from words to trees, the index hierarchy becomes strict, is particularly interesting.

REFERENCES

- [BC18] Mikołaj Bojańczyk and Wojciech Czerwiński. *Automata-Toolbox*. University of Warsaw, 2018. last checked April 2018.
- [BDM17] Massimo Benerecetti, Daniele Dell’Erba, and Fabio Mogavero. Robust exponential worst cases for divide-et-impera algorithms for parity games. *Proceedings of International Symposium on Games, Automata, Logics, and Formal Verification*, pages 121–135, 2017.
- [BGKR12] Dietmar Berwanger, Erich Grädel, Łukasz Kaiser, and Roman Rabinovich. Entanglement and the complexity of directed graphs. *Theoretical Computer Science*, 463:2–25, 2012.
- [BL18] U. Boker and K. Lehtinen. On the way to alternating weak automata. In *Proceedings of FSTTCS*, volume 122 of *LIPICs*, pages 21:1–21:22, 2018.
- [Bra98] J.C. Bradfield. The modal μ -calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195(2):133 – 153, 1998.
- [BS18] U. Boker and Y. Shaulian. Automaton-based criteria for membership in CTL. In *Proceedings of LICS*, pages 155–164, 2018.
- [BW18] Achim Blumensath and Felix Wolf. Bisimulation invariant monadic-second order logic in the finite. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- [CDF⁺19] Wojciech Czerwiński, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdziński, Ranko Lazić, and Paweł Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds

- for parity games. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2333–2349. SIAM, 2019.
- [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*, pages 252–263. ACM, 2017.
- [Col13] Thomas Colcombet. Fonctions régulières de coût. *Habilitation à diriger les recherches, École Doctorale de Sciences Mathématiques de Paris Centre*, 2013.
- [DDS12] S. Demri, A. K. Dhar, and A. Sangnier. Taming past LTL and flat counter systems. In *IJCAR*, pages 179–193, 2012.
- [DG08] Anuj Dawar and Erich Grädel. The descriptive complexity of parity games. In *Computer Science Logic*, pages 354–368. Springer, 2008.
- [EJ91] E Allen Emerson and Charanjit S Jutla. Tree automata, mu-calculus and determinacy. In *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*, pages 368–377. IEEE, 1991.
- [EP65] P. Erdős and L. Pósa. On independent circuits contained in a graph. *Canadian Journal of Mathematics*, 17:347–352, 1965.
- [FJS⁺17] John Fearnley, Sanjay Jain, Sven Schewe, Frank Stephan, and Dominik Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software*, pages 112–121. ACM, 2017.
- [Fri09] Oliver Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *Logic In Computer Science, 2009. LICS’09. 24th Annual IEEE Symposium on*, pages 145–156. IEEE, 2009.
- [Fri11] Oliver Friedmann. Recursive algorithm for parity games requires exponential time. *RAIRO-Theoretical Informatics and Applications*, 45(4):449–457, 2011.
- [HP06] Thomas A Henzinger and Nir Piterman. Solving games without determinization. In *International Workshop on Computer Science Logic*, pages 395–410. Springer, 2006.
- [JL17] Marcin Jurdziński and Ranko Lazic. Succinct progress measures for solving parity games. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, volume 00, pages 1–9, June 2017.
- [Jur98] Marcin Jurdziński. Deciding the winner in parity games is in $UP \cap co-UP$. *Information Processing Letters*, 68(3):119–124, 1998.
- [Jur00] Marcin Jurdziński. Small progress measures for solving parity games. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 290–301. Springer, 2000.
- [KF11] L. Kuhlmann and B. Finkbeiner. Weak kripke structures and LTL. In *CONCUR*, pages 419–433, 2011.
- [Kir02] D. Kirsten. *Alternating Tree Automata and Parity Games*, pages 153–167. Springer Berlin Heidelberg, 2002.
- [Koz88] Dexter Kozen. A finite model theorem for the propositional μ -calculus. *Studia Logica*, 47(3):233–241, 1988.
- [KV98] O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proceedings of STOC*, pages 224–233, 1998.
- [KV01] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(2):408–429, 2001.
- [Leh18] K. Lehtinen. A modal μ perspective on solving parity games in quasipolynomial time. In *Proceedings of LICS*, 2018.
- [Lin88] P. A. Lindsay. On alternating omega-automata. *J. Comput. Syst. Sci.*, 36(1):16–24, 1988.
- [McN66] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and control*, 9(5):521–530, 1966.
- [Mos91] A Mostowski. Games with forbidden positions. university of gdansk, gdansk. Technical report, Poland, Tech. Rep, 1991.
- [MS95] D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

- [SMPV16] A. Di Stasio, A. Murano, G. Perelli, and M. Y. Vardi. Solving parity games using an automata-based algorithm. In *International Conference on Implementation and Application of Automata*, pages 64–76. Springer, 2016.
- [Wil99] T. Wilke. CTL^+ is exponentially more succinct than CTL . In *Proceedings of FSTTCS*, volume 1738 of *LNCS*, pages 110–121. Springer, 1999.
- [Wil01] Thomas Wilke. Alternating tree automata, parity games, and modal μ -calculus. *Bulletin of the Belgian Mathematical Society Simon Stevin*, 8(2):359, 2001.