Parity Games of Bounded Tree-Depth

Konrad Staniszewski
University of Warsaw, Poland
IDEAS NCBR Sp. z o.o., Warsaw, Poland

Abstract

The exact complexity of solving parity games is a major open problem. Several authors have searched for efficient algorithms over specific classes of graphs. In particular, Obdržálek showed that for graphs of bounded tree-width or clique-width, the problem is in P, which was later improved by Ganardi, who showed that it is even in LOGCFL (with an additional assumption for clique-width case). Here we extend this line of research by showing that for graphs of bounded tree-depth the problem of solving parity games is in logspace uniform AC^0. We achieve this by first considering a parameter that we obtain from a modification of clique-width, which we call shallow clique-width. We subsequently provide a suitable reduction.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Parity Games, Circuits, Tree-Depth, Clique-Width

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.33

Related Version *Full Version*: `https://arxiv.org/abs/2211.02926` [28]

Funding *Konrad Staniszewski*: Work supported by the National Science Centre, Poland (grant no. 2021/41/B/ST6/03914).

Acknowledgements I want to thank the supervisor of my master's thesis Damian Niwiński and anonymous reviewers for valuable feedback.

1 Introduction

Parity games are two-player games played on directed graphs with ranked vertices $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V, rank : V \to \mathbb{N})$. Player X makes a move at vertices from V_{X} by moving a token along one of the outgoing edges. By making moves, players form a sequence of vertices called a play. A play may be finite or not, but it must be exhaustive. That is, whenever there is a possibility to make a move, then a move is made. Player E wins a play π if it is either finite and ends at a vertex from V_{O} or is infinite and the parity of the highest rank that occurs infinitely often is even, otherwise player O wins.

Parity games play an important role in system verification and synthesis [23]. However, the exact complexity of solving parity games is still an open problem. The fastest known algorithms work in time $n^{\mathcal{O}\left(\log\left(1+\frac{d}{\log(n)}\right)\right)}$ [22, 21], where n is the size of the graph and d is the number of different ranks. The problem was also tackled from the point of view of parameterized complexity. Obdržálek showed that in graphs with bounded tree-width [26] or clique-width [27] parity games can be solved in polynomial time. The result for tree-width was later improved to NC^2 by Fearnley and Schewe [13] and to LOGCFL by Ganardi [16]. The result for clique-width was improved by Ganardi to LOGCFL under an additional assumption that the k-expression is given [16]. It was also shown that parity games admit a polynomial-time algorithm on graphs with bounded entanglement [4], DAG-width [3], or Kelly-width [20] (for Kelly-width, decomposition is assumed to be a part of the input). For more about these classes see [17]. Calude, Jain, Khoussainov, Li, and Stephan in their breakthrough work about solving parity games in quasipolynomial time [5] showed that the problem is fixed-parameter tractable when parameterized by the number of ranks.

In this work, we show that solving parity games on graphs of bounded tree-depth is in logspace uniform AC^0 [1]. We start by introducing a parameter, which we call shallow clique-width, in Section 2. In Section 3, we introduce tools that are subsequently used in Section 4 to create an AC^0 algorithm for parity games on graphs of bounded shallow clique-width. In Section 5, we show how to efficiently reduce the problem of solving parity games on graphs of bounded tree-depth to solving parity games on graphs of bounded shallow clique-width. Our result improves over the ones of Obdržálek and Ganardi when considering graphs of bounded tree-depth. That is because logspace uniform AC^0 is a strict subclass of LOGCFL (parity is not in AC^0 [14, 1]), and graphs of bounded tree-depth have bounded tree-width. In addition to this, our intermediate result for shallow clique-width is an example of an AC^0 algorithm for solving parity games on a class of graphs that are not sparse (assuming an appropriate graph description is provided).

Related Work

The relation between uniform AC^0 and tree-depth was studied in [10, 11]. Elberfeld, Jakoby, and Tantau showed that MSO definable problems are in uniform AC^0 when considering structures whose Gaifman graphs have bounded tree-depth [11]. Elberfeld, Grohe, and Tantau showed that on graphs of bounded tree-depth FO, MSO, and GSO (guarded second-order logic) have the same expressive power [10]. The question of whether the winning regions of parity games are definable in MSO is unsolved for the case when the number of ranks is unbounded. Dawar and Grädel showed that they are not definable in μ-calculus (a fragment of MSO), but can be defined in GSO if we use a quasi-order over vertices (having a higher rank) along with a unary predicate OddRank [8]. However, this along with [10, 11] does not place parity games on graphs of bounded tree-depth in uniform AC^0 since the tree-depth of the Gaifman graphs of the structures considered in [8] is unbounded due to the aforementioned quasi-order.

We use a dynamic programming approach along the graph decomposition similar to the one of Obdržálek [27]. However, dealing with AC^0 requires more technical care. In particular, we need to provide definitions of operations that are computable in logspace uniform AC^0. Another difference is that we don't restrict our attention to t-strategies but to a potentially larger set of positional strategies. We also provide the efficient reduction from solving parity games on graphs of bounded tree-depth to solving parity games on graphs of bounded shallow clique-width.

2 Definitions and Notation

2.1 Basic Notation

We use $g = f[y \mapsto c]$ to define g that coincides with f except that $g(y) = c$. To define a partial function that coincides with f but is undefined on y we write $f[y \mapsto \text{undefined}]$. By $\text{Dom}(f)$ we denote the set of elements for which f is defined. Given a function $f : A \to B$ and finite or infinite sequence $s = a_0, a_1, \ldots$ of elements of A, we denote application of f to s by $f(s) = f(a_0), f(a_1), \ldots$.

[1] A problem is said to be in logspace uniform AC^0 if there is a constant k, a polynomial p, and a deterministic Turing machine that given an input size N, generates in space $\mathcal{O}(\log(N))$ the description of a circuit of size $\mathcal{O}(p(N))$ and depth at most k that solves the problem instances of size N. The circuit can be viewed as the directed graph with vertices of in-degree 0 denoted as input, out-degree 0 as output, and other vertices labeled with boolean operations ($\wedge$, $\vee$ or $\neg$; both $\wedge$ and $\vee$ can take an arbitrary number of arguments). Edges show computation flow. In such graphs, vertices are usually called gates. For a more detailed definition, see [2].

2.2 Arenas, Games, Colorings

An *arena* is a directed graph $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V)$ with vertices partitioned between players E and O. In a parity game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V, rank : V \to \mathbb{N})$, the behavior of player X is modeled by a partial function $\rho_{\mathrm{X}} : V^* \times V_{\mathrm{X}} \rightharpoonup V$ called *strategy*, which given the sequence of visited vertices and the vertex that the token is on, gives the next vertex to move the token to. Every strategy must be exhaustive and correct – whenever there is a possibility of player movement, then the function must be defined, and the token can only be moved along existing edges. A strategy ρ_{X} is *positional* if the result depends only on the vertex the token is on (i.e. for a vertex $v \in V_{\mathrm{X}}$ and a prefix of a play $\vec{\pi}$ we have $\rho_{\mathrm{X}}(\vec{\pi}, v) = \rho_{\mathrm{X}}(v)$). A play π is consistent with ρ_{X} if whenever it is the turn of player X, then the move is made according to ρ_{X}. A strategy ρ_{X} is winning for player X from a vertex v if all plays starting at v and consistent with ρ_{X} are winning for player X. It is known that parity games are globally positionally determined [12]. That is for $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank)$ there exists a partition of V into the winning regions W_{E} and W_{O} such that E has a positional strategy ρ_{E} that wins from every vertex in W_{E}, and O has a positional strategy ρ_{O} that wins from every vertex in W_{O}.

We define $\Pi_G(\rho_{\mathrm{X}}, v)$ as the set of plays in the arena G that start at the vertex v and are consistent with the strategy ρ_{X} for player X, whereas by $\vec{\Pi}_G(\rho_{\mathrm{X}}, v, w)$ we define their prefixes that end at w (i.e. $\vec{\Pi}_G(\rho_{\mathrm{X}}, v, w) = \{\pi[\dots i] : \pi[i] = w \wedge \pi \in \Pi_G(\rho_{\mathrm{X}}, v)\}$). To distinguish a play from its prefix, we denote plays using the symbol π and prefixes of plays using $\vec{\pi}$.

We say that $Color : V \to C_{\mathrm{E}} \uplus C_{\mathrm{O}}$ is a player-aware coloring of vertices from $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E)$ if vertices from V_X are colored using colors from C_{X}. We will sometimes write a parity game along with player-aware coloring of its arena and say that $G' = (V' = V'_{\mathrm{E}} \uplus V'_{\mathrm{O}}, E', rank', Color')$ is a subgame of $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank, Color)$ if and only if $V'_{\mathrm{E}} \subseteq V_{\mathrm{E}}$, $V'_{\mathrm{O}} \subseteq V_{\mathrm{O}}$, $E' \subseteq E$, $rank'$ is $rank$ with domain restricted to V' and $Color'$ is $Color$ with domain restricted to V'.

2.3 Shallow Clique-Width for Arenas

Here we define shallow clique-width for arenas, which can be viewed as clique-width [6, 7] with unions of arbitrary number of components (instead of only binary ones) and restricted term height.

▶ **Definition 1.** *A* tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ *consists of a rooted tree* T*, with all root-to-leaf paths having the same length, a set of colors* $C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}$*, a coloring* $Color$ *of leaves of* T*, and a function* D *that assigns to each internal node of* T *a subset of* $C \times C$*.*

For a tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ and a node l of T, by $\mathcal{TM}_l$ we mean the tree-model $(T_l, C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color', D')$ where T_l is the subtree of T rooted at l, $Color'$ is $Color$ with domain restricted to leaves of T_l, and D' is D with domain restricted to internal nodes of T_l.

▶ **Definition 2.** *We define an arena* $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V)$ *induced by* $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ *in the following way.*

- *If* T *consists of one node* v *and* $Color(v) \in C_{\mathrm{X}}$*, then it induces edgeless* G *with* $V_{\mathrm{X}} = \{v\}$*.*
- *Otherwise, let* l *be* T*'s root and let* $l_1, \dots l_m$ *be children of* l*. The arena induced by* $\mathcal{TM}$ *is created by taking union of arenas induced by* $\mathcal{TM}_{l_1}, \dots, \mathcal{TM}_{l_m}$*, and adding a directed edge from every vertex colored* c *to every vertex colored* c' *whenever* $\langle c, c' \rangle \in D(l)$*.*

For an example of a tree-model and the induced arena see Figure 1.

▶ **Definition 3.** *The* shallow clique-width *of an arena $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V)$ is the smallest k such that there exists a tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ with $|C| \leq k$ and T of height at most k such that the induced arena is G.*

One might observe that shallow clique-width is similar to the notion of shrub-depth [19, 18]. The main difference, however, is that for a shallow clique-width's tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ and an internal node l of T a pair $\langle c, c' \rangle \in D(l)$ affects all pairs $\langle v, v' \rangle$ of leaves in the subtree of T rooted at l such that $Color(v) = c$ and $Color(v') = c'$, whereas in shrub-depth it is additionally restricted to only those for which the lowest common ancestor in T is l.

It is worth mentioning that if a class of graphs has bounded shallow clique-width, then it has bounded clique-width and shrub-depth. The former observation follows from the definition of clique-width, the latter one from the fact that in the shallow clique-width's tree-model there are at most $2^{|C \times C|}$ sets $D(l)$ at each level of the tree-model. So at the expense of an increase in the number of colors, we can get rid of these sets and describe connections using only the distance to the lowest common ancestor and colors.

2.4 Tree-Depth

Tree-depth was originally defined for undirected graphs [24]. Here we adapt this definition to arenas by simply forgetting the orientation of edges.

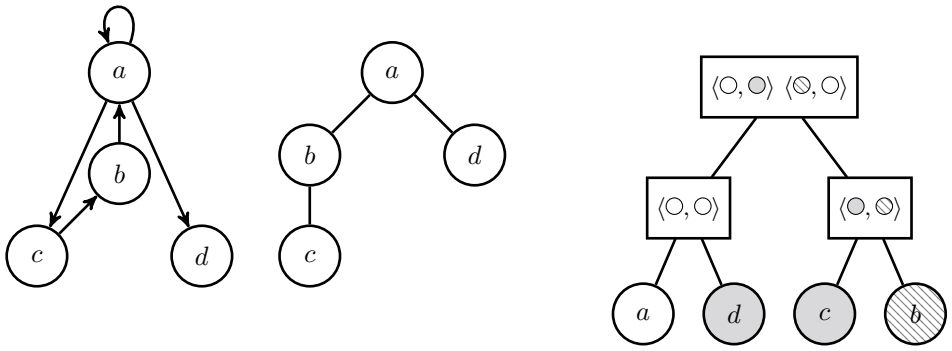
▶ **Definition 4.** *A forest of rooted trees $\mathcal{F}$ is an* elimination forest *of an arena $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V)$ if vertices of $\mathcal{F}$ are vertices of G and $\langle v, w \rangle \in E$ implies that v and w are in the same tree T in $\mathcal{F}$ and either v is an ancestor of w or w is an ancestor of v in T (see Figure 1). Tree-depth of an arena G is the smallest height of an elimination forest of G.*

▶ **Lemma 5** ([25, Chapter 6.2])**.** *If an arena G has an elimination forest of height k, then any simple path in G has length at most $2^{k+1} - 2$.*

The lemma above can be proven by induction on the height of the elimination forest.

One can think of graphs of bounded tree-depth as graphs of bounded tree-width [9, Chapter 12] with an additional constraint on the height of the tree-decomposition. More precisely, if a graph G has a tree-decomposition of height h where each bag contains at most k vertices, then it has tree-depth at most $k(h+1)$ [25, Chapter 6.4] (to obtain the elimination forest from the tree-decomposition, we leave each vertex of G in the bag that is closest to the root of the tree-decomposition and then replace each bag in the tree-decomposition with a path of its elements). What is more, if a graph has tree-depth k, then it admits a tree-decomposition of height bounded by $k+1$ where each bag contains at most $k+2$ vertices (to obtain the tree-decomposition, we take the elimination forest $\mathcal{F}$, make it a tree T by choosing a root r of an arbitrary tree in $\mathcal{F}$ and connecting other roots to r, and define bags for vertices of T as sets of vertices on vertex-r paths).

Shallow clique-width is more general than tree-depth. First, for an arena G of tree-depth k one can use $2(k+1)3^{k+1}$ colors to create a tree-model of height $k+2$ that induces G. This is because we can take some elimination forest $\mathcal{F}$ of G of height k and color each vertex according to its player-membership in G, and depth and connections to ancestors in $\mathcal{F}$ (for each vertex v and its ancestor w we have that either there is a directed edge from v to w, to v from w, or v and w are not connected in G). On the other hand, arenas where each vertex has a directed edge to every vertex have small shallow clique-width, but large tree-depth.



(a) Arena **(b)** Elimination forest **(c)** Tree-model

Figure 1 Arena (a), an elimination forest for the arena (b) and a tree-model that induces the arena (c) (all vertices belong to one player).

The major advantage of tree-depth over shallow clique-width is that given an arena G of tree-depth at most k, we know how to efficiently find an elimination forest of G of height at most k (details in Section 5.4). It is also worth noting that despite the simplicity of tree-depth, some problems are $W[1]$-*hard* when parameterized by it [15, Lemma 2, Proposition 5].

As mentioned before, in this work, we take a detour by first proving that solving parity games played on arenas of bounded shallow clique-width is in logspace uniform AC^0 (assuming an appropriate graph description is provided) and then providing a reduction. We decided to do so as thinking about vertices of the same color as the ones that will be treated similarly turned out to be conceptually simpler from our perspective. What is more, such an approach gave us a logspace uniform family of AC^0-circuits for a class of graphs that are not sparse.

3 Tools

Now we proceed with tools for constructing a logspace uniform family of circuits for parity games played on arenas with bounded shallow clique-width. The approach presented here is similar to the one used in [27] for creating a polynomial-time algorithm for parity games played on arenas of bounded clique-width. Here we also operate on some succinct information characterizing strategies that we call enforcements. Enforcements are an adaptation of Obdržálek's borders [26] from tree-width to shallow clique-width. The main idea behind borders is to store for a separating set S, a strategy ρ_{E}, and a vertex v information about the worst arrivals at vertices of S that opponent can achieve when starting from v and playing against ρ_{E}, whereas in enforcements we are interested in knowing what the worst arrivals at vertices of a specific color can be. Our adaptation of borders is analogous to the one considered in [27].

We additionally extend and modify this approach to obtain a logspace uniform family of constant-depth circuits.

3.1 Parity Games With Stop

▶ **Definition 6.** *We define a* stop parity game *as a parity game where player* E *has an additional move called* STOP *at his positions. Executing this move ends the game in a draw.*

▶ **Observation 7.** *In an arena $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V)$ with vertices ranked by $rank : V \to \mathbb{N}$, winning regions for player* E *in both stop and normal parity games are equal. What is more if a strategy ρ_{E} is winning in either of the games, then it is winning in the other one, as ρ_{E} cannot be forced to use STOP.*

▶ **Definition 8.** *We say that a strategy ρ_{E} is* safe from *v in a stop parity game G, if all plays that start at v and are consistent with ρ_{E} are not losing for player* E.

Stop parity games will turn out to be useful for solving parity games on arenas that can be extended.

3.2 Enforcements

▶ **Definition 9.** *Let* $\mathrm{elem}(s)$ *denote the set of elements that appear in the sequence s. For a game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank)$, we define a function $mr_G(\vec{\pi}) = \max(\mathrm{elem}(rank(\vec{\pi})))$. That is for a prefix of a play, return the maximum rank of its vertex.*

▶ **Definition 10.** *We define a total order $\prec$ on natural numbers as $p \prec q \Leftrightarrow (-1)^p p < (-1)^q q$ and $p \preceq q \Leftrightarrow (p \prec q \vee p = q)$. That is $\ldots \prec 5 \prec 3 \prec 1 \prec 0 \prec 2 \prec 4 \prec 6 \prec \ldots$.*

▶ **Definition 11.** *For a stop parity game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank)$, a strategy ρ_{E} and $v \in V$ we define a partial function*

$$\mathrm{venf}^G_{\rho_{\mathrm{E}},v}(w) = \begin{cases} \min_{\preceq}\{mr_G(\vec{\pi}) : \vec{\pi} \in \vec{\Pi}_G(\rho_{\mathrm{E}}, v, w) \wedge \rho_{\mathrm{E}}(\vec{\pi}) = \mathit{STOP}\} & \text{if } w \in V_{\mathrm{E}}. \\ \min_{\preceq}\{mr_G(\vec{\pi}) : \vec{\pi} \in \vec{\Pi}_G(\rho_{\mathrm{E}}, v, w)\} & \text{if } w \in V_{\mathrm{O}} \end{cases}$$

Where $\min_{\preceq}(A)$ is a partial function that for a finite subset of natural numbers A gives the smallest element of A according to $\preceq$ and is undefined for $A = \emptyset$.

Intuitively for $w \in V_{\mathrm{O}}$ the operation above returns the worst possible arrival at w that player O can force when playing against the strategy ρ_{E} and for $w \in V_{\mathrm{E}}$ the worst arrival at w such that ρ_{E} will decide to stop the game.

▶ **Definition 12.** *We naturally adapt the definition above to games $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank)$ with player-aware coloring $Color : V \to C_{\mathrm{E}} \uplus C_{\mathrm{O}}$:*

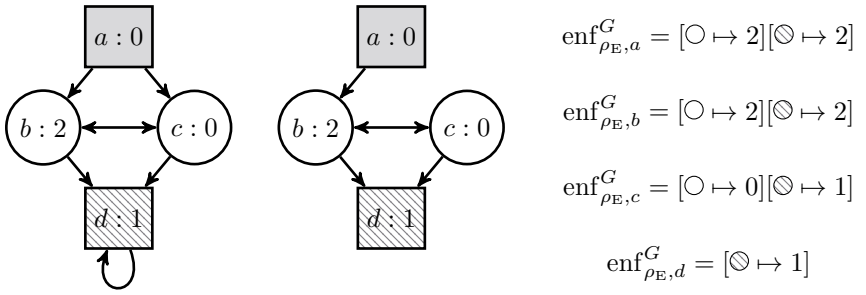
$$\mathrm{enf}^G_{\rho_{\mathrm{E}},v}(c) = \min_{\preceq}\{\mathrm{venf}^G_{\rho_{\mathrm{E}},v}(w) : w \in \mathrm{Dom}(\mathrm{venf}^G_{\rho_{\mathrm{E}},v}) \wedge Color(w) = c\}$$

An *enforcement* is any partial function from C to $\{0, \ldots, \max(rank(V))\}$. From now on by *enforcement from v* of a strategy ρ_{E} in G we will mean $\mathrm{enf}^G_{\rho_{\mathrm{E}},v}$. Note that this is always an enforcement and that ρ_{E} can have different enforcements from different vertices (see Figure 2).

▶ **Observation 13.** *Enforcement $\mathrm{enf}^G_{\rho_{\mathrm{E}},v}$ is defined for some color of player* E *(i.e. color c such that $c \in C_{\mathrm{E}}$) if and only if there is a finite play π consistent with ρ_{E} that starts at v such that $\rho_{\mathrm{E}}(\pi) = \mathit{STOP}$.*

▶ **Definition 14.** *We define a partial order $\sqsubseteq$ on enforcements as follows:*
$P \sqsubseteq Q \Leftrightarrow \mathrm{Dom}(P) \subseteq \mathrm{Dom}(Q) \wedge \forall a \in \mathrm{Dom}(P). Q(a) \preceq P(a)$
$P \sqsubset Q \Leftrightarrow P \sqsubseteq Q \wedge P \neq Q$.

Intuitively $P \sqsubset Q$ means that Q describes worse arrival scenarios from player E perspective than P.



(a) Parity game. **(b)** Parity game restricted to ρ_E.

Figure 2 A parity game G (a), G with the moves of player E restricted to the positional strategy $\rho_E = [\langle \overrightarrow{\pi}, a\rangle \mapsto b][\langle \overrightarrow{\pi}, d\rangle \mapsto \mathit{STOP}]$ along with the enforcements of ρ_E from various vertices (b). Square vertices belong to player E, round ones to O. The letters denote vertices, whereas the numbers inside vertices denote their ranks. Background shading denotes vertex color (vertices b and c have the same color, whereas colors of a, b and d are pairwise distinct).

Operations on Enforcements

Clearly, modifications of a strategy ρ_E can result in modifications of enforcements of this strategy. What is more, some of these modifications can be done directly on the enforcements without knowing the original strategy. In the further parts of this work, we will be interested in modifications that adapt strategies of player E from an arena G to an arena G', that was made from G by addition of directed edges between vertices of specified colors. It will usually suffice to remember enforcements of a strategy instead of the strategy itself and to perform respective operations on enforcements. Below we introduce some useful operations on enforcements.

The first operation corresponds to a modification of a strategy ρ_E to the strategy ρ'_E that behaves like ρ_E, but each time ρ_E decides to STOP at some vertex of a specified color c ρ'_E uses the option to move to the vertex from which the game was started. Moreover, we want to ensure that ρ'_E will never close a cycle with the highest rank being odd by doing so.

▶ **Definition 15.** *For $G = (V = V_E \uplus V_O, E \subseteq V \times V, rank : V \to \mathbb{N}, Color : V \to C_E \uplus C_O)$, enforcement P and color c we define*

$$\mathrm{loop}(P,c) = \begin{cases} P & \text{if } c \in C_E \wedge c \notin \mathrm{Dom}(P) \\ P[c \mapsto \mathit{undefined}] & \text{if } c \in C_E \wedge P(c) \text{ is even} \end{cases}$$

Please note that $\mathrm{loop}(P,c)$ is undefined if $c \in C_O$ or $P(c)$ is odd. Let P be the enforcement of ρ_E from some vertex v. Let c be any element of C_E and assume that from each vertex of color c there is an edge to v. Now, the first case in the definition of $\mathrm{loop}(P,c)$ corresponds to the situation when ρ_E will never stop at a vertex of color c when starting the game from v. Whereas the second one to the situation when ρ_E can be forced to stop at a vertex of color c, but the prefix of the play that will lead here will always have even max rank.

▶ **Definition 16.** *For $G = (V = V_E \uplus V_O, E \subseteq V \times V, rank : V \to \mathbb{N}, Color : V \to C_E \uplus C_O)$, $r \in \mathbb{N}$, color c and enforcements P, Q, we define two partial functions*

$$\mathrm{lift}(Q,r)(c) = \max(Q(c), r) \text{ if } c \in \mathrm{Dom}(Q)$$
$$\widetilde{\min}(P,Q)(c) = \min_{\preceq}(P(c), Q(c))$$

In this writing, we allow one of $\min_{\preceq}(P(c), Q(c))$ *arguments to be undefined and define* $\min_{\preceq}(P(c), Q(c))$ *to be the defined one (or undefined when both are undefined). The* $\max(Q(c), r)$ *in the definition of* $\mathrm{lift}(Q, r)(c)$ *is taken using the standard order on* $\mathbb{N}$.

Another operation is for composing strategies in a way that corresponds to playing one strategy and switching to another when either reaching vertices of specified color $c \in C_{\mathrm{O}}$ or stopping at vertices of specified color $c \in C_{\mathrm{E}}$.

▶ **Definition 17.** *For* $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V, rank : V \to \mathbb{N}, Color : V \to C_{\mathrm{E}} \uplus C_{\mathrm{O}})$, *color* c *and enforcements* P, Q, *we define*

$$\mathrm{merge}(P, c, Q) = \begin{cases} \widetilde{\min}(P, \mathrm{lift}(Q, P(c))) & \text{if } c \in \mathrm{Dom}(P) \wedge c \in C_{\mathrm{O}} \\ \widetilde{\min}(P[c \mapsto \textit{undefined}], \mathrm{lift}(Q, P(c))) & \text{if } c \in \mathrm{Dom}(P) \wedge c \in C_{\mathrm{E}} \\ P & \text{otherwise} \end{cases}$$

Intuitively, taking $\mathrm{lift}(Q, P(c))$ in the formula above corresponds to extending plays that were used to calculate P with the ones being used to calculate Q. More detailed explanation is the following.

▶ **Lemma 18.** *Properties of* $\mathrm{merge}(P, m, Q)$ *and* $\mathrm{loop}(P, m)$*:*

1. *Let* P, P', Q, Q' *be enforcements such that* $P' \sqsubseteq P$ *and* $Q' \sqsubseteq Q$, *then*

$$\mathrm{merge}(P', c, Q') \sqsubseteq \mathrm{merge}(P, c, Q)$$

2. *Let* c *be a color of player* O. *Let* ρ_{E}, ρ'_{E}, *be such that* ρ_{E} *is safe from* v, *whereas* ρ'_{E} *is safe from every vertex of color* c *in* G. *Let* ρ''_{E} *be like* ρ_{E}, *but when* ρ_{E} *reaches a vertex of color* c, ρ''_{E} *forgets about the past and switches permanently to* ρ'_{E} *behavior. Then* ρ''_{E} *is safe from* v *in* G *and*

$$\mathrm{enf}^G_{\rho''_{\mathrm{E}}, v} \sqsubseteq \mathrm{merge}(\mathrm{enf}^G_{\rho_{\mathrm{E}}, v}, c, \widetilde{\min}(\mathrm{enf}^G_{\rho'_{\mathrm{E}}, w_1}, \ldots, \mathrm{enf}^G_{\rho'_{\mathrm{E}}, w_m}))$$

   where $w_1, \ldots, w_m$ *are all vertices of color* c *and* $\widetilde{\min}(A, B, C) = \widetilde{\min}(\widetilde{\min}(A, B), C)$.
3. *Let* ρ_{E}, ρ'_{E}, *be strategies that are safe from* v *and* w *in* G *respectively. Let* c *be a color of player* E *such that from every vertex of color* c *there is an edge to* w, *and let* ρ''_{E} *be like* ρ_{E}, *but when* ρ_{E} *decides to stop at some vertex of color* c, ρ''_{E} *moves to* w *instead, forgets about the past and switches permanently to* ρ'_{E} *behavior. Then* ρ''_{E} *is safe from* v *in* G *and*

$$\mathrm{enf}^G_{\rho''_{\mathrm{E}}, v} = \mathrm{merge}(\mathrm{enf}^G_{\rho_{\mathrm{E}}, v}, c, \mathrm{enf}^G_{\rho'_{\mathrm{E}}, w})$$

4. *Let* ρ_{E} *be a strategy that is safe from* v *in* G. *Let* c *be a color of player* E *such that from every vertex of color* c *there is an edge to* v, *and let* ρ'_{E} *be like* ρ_{E}, *but each time* ρ_{E} *decides to stop at some vertex of color* c, ρ'_{E} *moves to* v, *forgets about the past and resumes as* ρ'_{E}. *Then if* $\mathrm{loop}(\mathrm{enf}^G_{\rho_{\mathrm{E}}, v}, c)$ *is defined, we have that* ρ'_{E} *is safe from* v *in* G *and*

$$\mathrm{enf}^G_{\rho'_{\mathrm{E}}, v} = \mathrm{loop}(\mathrm{enf}^G_{\rho_{\mathrm{E}}, v}, c)$$

The lemma above can be proven by simple but tedious case analysis. The key step in the proof is to show that $p \preceq p' \wedge q \preceq q'$ implies $\max(p, q) \preceq \max(p', q')$, what can be done by considering cases on the parity of p' and q'.

Enforcements and Strategies

For a set of enforcements $\mathcal{E}$ and a vertex v of a game G we consider the following properties.

1. (*soundness*) If $P \in \mathcal{E}$, then there exists a (possibly non-positional) strategy ρ_{E} that is safe from v in G such that $\mathrm{enf}^{G}_{\rho_{\mathrm{E}},v} \sqsubseteq P$.
2. (*completeness*) For each positional strategy ρ_{E} that is safe from v in G we have that $\mathrm{enf}^{G}_{\rho_{\mathrm{E}},v} \in \mathcal{E}$.
3. (*up-closure*) If $P \in \mathcal{E}$ and $P \sqsubseteq Q$ then $Q \in \mathcal{E}$.

We will say that $\mathcal{E}$ is *valid* for v if it is sound, complete and up-closed for v.

From now on we will assume that for a stop parity game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank)$ we always have $rank(V) \subseteq \{0, \ldots, 2|rank(V)| - 1\}$, and denote $2|rank(V)|$ by d, as otherwise we can sort elements of $rank(V)$ and put them in the desired range.

▶ **Lemma 19.** *Given a set $\mathcal{E}$ of enforcements in G that is valid for v, we have that player* E *wins the parity game on G's arena from v if and only if $\mathcal{E}$ contains an enforcement that is undefined for every color of player* E.

Proof. If player E wins from v in the parity game played on G's arena, then there exists a positional strategy ρ_{E} that wins the parity game starting at v, and the enforcement generated by ρ_{E} from v in G is undefined for every color of player E (see Observation 13). What is more, since $\mathcal{E}$ is valid for v, from Observation 7 we have that $\mathrm{enf}^{G}_{\rho_{\mathrm{E}},v} \in \mathcal{E}$. On the other hand, if $\mathcal{E}$ contains an enforcement P that is undefined for every color of player E, then since $\mathcal{E}$ is valid for v, there exists a strategy ρ_{E} that is safe from v such that $\mathrm{enf}^{G}_{\rho_{\mathrm{E}},v} \sqsubseteq P$, what by Observation 13 means that ρ_{E} never uses STOP. ◀

In the following parts, we will aim to calculate for each vertex of a game a valid set of enforcements.

4 Circuit Family for Shallow Clique-Width

Now we will show how for $n, d > 1$, and $k > 0$ create in space $\mathcal{O}(\log(n) + \log(d)k)$ (and therefore time $n^{\mathcal{O}(1)}d^{\mathcal{O}(k)}$) the circuit $\mathbb{SCW}_{\langle n,d,k\rangle}$ of depth $\mathcal{O}(k^3)$, that given the specific representation of a parity game of shallow clique-width at most k, computes the partition into winning regions.

4.1 Overview

Circuit $\mathbb{SCW}_{\langle n,d,k\rangle}$ will be a combination of auxiliary circuits called gadgets. The circuit will input parity game's G arena as a tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ (details about the input are presented in Section 4.2), and it will process $\mathcal{TM}$ bottom-up.

The first layers of the circuit will calculate valid sets of enforcements for one-vertex arenas consisting of leaves of T. To do so gadgets defined in Section 4.3.1 will be used. The following layers of the circuit will correspond to computing valid sets of enforcements for larger and larger arenas induced by larger and larger subtrees of the given tree-model. For the last layers, we will use Lemma 19 to get the partition into winning regions.

To be more precise for an internal node l of T the circuit will consider a subgame G_l played on the arena induced by $\mathcal{TM}_l$. For each vertex of such G_l the circuit will calculate a valid set of enforcements using valid sets of enforcements calculated for subgames played on arenas induced by children of l. This calculation will be divided into several steps. First, a disjoint union of arenas induced by children of l will be taken, and then connections described

by $D(l)$ will be added one by one. For a pair $\langle s,t\rangle \in D(l)$ cases on whether s is a color of player E or O will be considered. Let G' be the subgame of G_l just before processing of $\langle s,t\rangle$ and G'' be G' after adding directed edges from all vertices of color s to all of color t in G'.

For the case when s is a color of player E, we will observe that a positional strategy ρ_{E} that is safe from a vertex v in G'' can be modified to a strategy ρ'_{E} such that ρ'_{E} is also safe from v, $\mathrm{enf}^{G''}_{\rho'_{\mathrm{E}},v} \sqsubseteq \mathrm{enf}^{G''}_{\rho_{\mathrm{E}},v}$ and ρ'_{E} behaves similarly to ρ_{E} but when ρ_{E} decides to move along a newly added edge, then ρ'_{E} chooses to move to some fixed vertex w. Given such ρ'_{E} we will be able to factor it into two strategies that are safe in G' from v and w respectively.

For the case when s is a color of player O, we will observe that a positional strategy of player E that is safe from v in G'' can be factored into a positional strategy that is safe from v in G' and a positional strategy that is safe from all vertices of color t in G'.

These observations will allow us to calculate valid sets of enforcements for vertices of G'' using valid sets of enforcements calculated for G'. These updates will be performed using gadgets defined in Section 4.3.3. The whole procedure of adding connections between vertices of specified colors will be handled by the gadget defined in Section 4.3.5. The details about connections between gadgets and therefore the construction of the whole circuit are presented in Section 4.4.

It is worth noting that the approach presented in this section fails for parity games of bounded shrub-depth. In short, that is because it makes use of the fact that in shallow clique-width, we specify connections using colors, whereas in shrub-depth, we additionally condition connections on the lowest common ancestor in the tree-model. To be more precise, when constructing an arena bottom-up from shallow clique-width's tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ during the processing of an internal vertex l of T for $\langle s,t\rangle \in D(l)$, we have that there is an edge from each leaf of T_l of color s to each leaf of T_l of color t. In shrub-depth, this is additionally conditioned on whether l is the lowest common ancestor of the vertices to be connected. This means that in shrub-depth a vertex is distinguished not only by its color but also by the information which subtree of T_l has the vertex as a leaf. Simple attempts to mitigate that either increase the depth of the circuit to be logarithmic or the number of possible enforcements to be exponential in the number of vertices.

4.2 Circuit Input

The circuit $\mathbb{SCW}_{\langle n,d,k\rangle}$ inputs an n-vertex game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V, rank : V \to \mathbb{N}, Color : V \to C_{\mathrm{E}} \uplus C_{\mathrm{O}})$ as $rank$ and a tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ such that $rank(V) \subseteq \{0, \dots d-1\}$, $d \leq 2n$, $|C| = k$, the height of T is k and $\mathcal{TM}$ induces the arena of G. The whole input is encoded as follows.

- For each vertex $v \in V$: k bits that one-hot encode its color (denoted as $COLOR_v$, one bit for one color, only the bit denoting the true color of the vertex is going to be true), and d bits that one-hot encode $rank(v)$ (denoted as $RANK_v$).
- k bits for color-to-player mapping: i'th bit is set to true if i'th color belongs to player E (denoted as $ECOLOR$).
- Description of T viewed as a subgraph of the graph consisting of $k+1$ layers, where each layer consists of n vertices, and each of them is connected to all vertices from the layer below (see Figure 3). So, for each $l \in \{1, \dots, k\}$ and $i \in \{1, \dots, n\}$:
  - n bits describing the connections to the layer below. (denoted as $CHILD_{l,i}$, for $l = 1$ it describes connections to layer 0, which is formed of vertices from V)
  - k^2 bits describing the result of D for a given internal node – one bit for each pair of colors (denoted as $DB_{l,i}$).

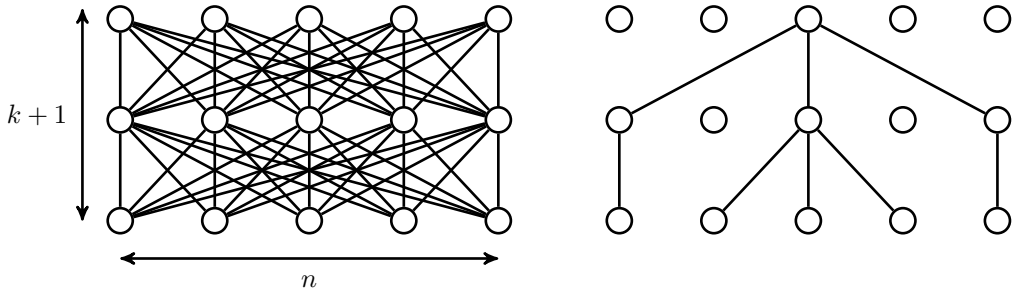


Figure 3 Tree embedding graph (left) and embedding of some tree (right).

.

We will call a distinguished collection of bits (which can be input bits and gate results) a bit pack. To denote a specific bit of a bit pack we will use square brackets notation. For example $COLOR_v[c]$ denotes the bit that is true if and only if vertex v has color c.

For validating input data, one can easily compute in space $\mathcal{O}(\log(n) + \log(k))$ parts of the circuit that have constant-depth and validate one-hot encodings. To validate T, one can compute in space $\mathcal{O}(\log(n) + \log(k))$ constant depth parts for validating that: each node has at most one parent, each node from layer 0 has exactly one parent, each node that has at least one child and is not from the last layer has a parent, each node that has a parent and is not from the layer 0 has at least one child, there is exactly one node with children in the last layer.

4.3 Gadgets

Here we define gadgets (auxiliary circuits) that will be used to construct the whole circuit.

4.3.1 Initialization

Here we define a constant-depth gadget that for a vertex v of G calculates a valid set of enforcements for v in an edgeless subgame of G that contains only one vertex v. We define the gadget so that it outputs empty sets of enforcements for other vertices of G and marks the vertex v in its output.

▶ **Definition 20.** INIT_v

$$A^{\text{out}}[w] = (w = v)$$
$$ENF_w^{\text{out}}[P] = (w = v) \wedge \bigvee_{\langle c,r\rangle : P \text{ is in up-closure of } \{[c \mapsto r]\}} COLOR_w[c] \wedge RANK_w[r]$$

Note that for every P the set $\{\langle c, r\rangle : P \text{ is in upward closure of } \{[c \mapsto r]\}\}$ can be enumerated in space $\mathcal{O}(\log(d)k)$. Note also that $\{P : P \text{ is in upward closure of } \{[c \mapsto r]\}\}$ is a valid enforcement set for a vertex v in one vertex game G where v has rank r and color c.

4.3.2 Combination Gadget

We define a constant-depth gadget that given m packs of input bits, each of length l (i'th bit of j'th pack denoted as $IN_j[i]$), and m bits that encode the choice of the bit pack (j'th bit denoted as $COND[j]$), outputs the result of the or operation on chosen bit packs.

▶ **Definition 21.** COMBINE^{m,l}

$$OUT[i] = \bigvee_{1 \le j \le m} IN_j[i] \wedge COND[j]$$

4.3.3 Update Gadget

Here we define a constant depth gadget that given for each vertex v of a subgame G' of G a valid set of enforcements outputs a valid set of enforcements for each vertex of the game G'' created from G' by adding directed edges from vertices of color s to those of color t.

Let $G' = (V' = V'_{\mathrm{E}} \uplus V'_{\mathrm{O}}, E', rank', Color')$ be any subgame of G. For a pair of colors $\langle s, t\rangle$ we define a gadget $\mathbb{U}^{\langle s,t\rangle}$ that inputs one bit $ECOLOR[s]$, and for each vertex v of G one bit encoding whether $v \in V'$ (denoted as $A^{\mathrm{in}}[v]$), $COLOR_v$, and $(d+1)^k$ bits denoted as ENF^{in}_v. For a vertex v of G', ENF^{in}_v will encode a set of enforcements that is valid for v in G' (one bit for one partial function from C to $\{0, \ldots, d-1\}$; bit for enforcement P denoted as $ENF^{\mathrm{in}}_v[P]$). We define $\mathbb{U}^{\langle s,t\rangle}$'s output to be $(d+1)^k$ bits for each vertex v of G (denoted as ENF^{out}_v). For a vertex $v \in V'$ those bits will encode enforcement set that is valid for v in the game G'' created from G' by adding connections from vertices colored s to ones colored t.

We consider two cases for a pair of colors $\langle s, t\rangle$. The first one is when color s belongs to player E (i.e. $ECOLOR[s]$ is true) and the other is when it belongs to player O. For the first case, we use gadget $\mathrm{U}^{s,t}_{\mathrm{E}}$, whereas for the second one $\mathrm{U}^{s,t}_{\mathrm{O}}$, both of which are defined below.

▶ **Definition 22.** $\mathrm{U}^{s,t}_{\mathrm{E}}$

$$MOVE[P] = \bigvee_{w} A^{\mathrm{in}}[w] \wedge COLOR_w[t] \wedge ENF^{\mathrm{in}}_w[P]$$

$$LOOPED[P] = s \notin \mathrm{Dom}(P) \wedge \bigvee_{2r \in \{0,\ldots,d-1\}} MOVE[P[s \mapsto 2r]]$$

$$OPT[P] = MOVE[P] \vee LOOPED[P]$$

$$ENF^{\mathrm{out}}_v[P] = A^{\mathrm{in}}[v] \wedge \left(\bigvee_{S \sqsubseteq P} ENF^{\mathrm{in}}_v[S] \vee \bigvee_{\langle Q,R\rangle : \mathrm{merge}(Q,s,R) \sqsubseteq P} ENF^{\mathrm{in}}_v[Q] \wedge OPT[R] \right)$$

Where $\mathrm{merge}(Q, s, R)$ *is calculated as if* s *was the color of player* E.

The intuition behind $\mathrm{U}^{s,t}_{\mathrm{E}}$ is that we should be able to modify each positional strategy ρ_{E} that is safe from v in game G'' to a strategy ρ'_{E} that behaves similarly to ρ_{E} but when ρ_{E} decides to make a move along a new edge, then ρ'_{E} always chooses to move to a fixed vertex w of color t. What is more, we should be able to make this modification in a way that $\mathrm{enf}^{G''}_{\rho'_{\mathrm{E}},v} \sqsubseteq \mathrm{enf}^{G''}_{\rho_{\mathrm{E}},v}$ and ρ'_{E} is also safe from v. In $ENF^{\mathrm{out}}_v[P]$ we check for the existence of such ρ'_{E} with $\mathrm{enf}^{G''}_{\rho'_{\mathrm{E}},v} \sqsubseteq P$. We do it by splitting ρ'_{E} into two parts: part till the new move (with enforcement $\sqsubseteq Q$) and after (with enforcement $\sqsubseteq R$). Note that ρ'_{E} defined as above will either make a move along a new edge at most once ($MOVE$) or an unbounded number of times ($LOOPED$).

▶ **Definition 23.** $\mathrm{U}^{s,t}_{\mathrm{O}}$

First, for each pair of enforcements Q, P, *and a vertex* v *such that* $Q \sqsubseteq P$*:*

$$\mathrm{SAFE}^{Q,P}_v = \neg A^{\mathrm{in}}[v] \vee \neg COLOR_v[t] \vee \bigvee_{R : \mathrm{eon}(R,s) \wedge \mathrm{merge}(Q,s,R) \sqsubseteq P} ENF^{\mathrm{in}}_v[R]$$

$$\mathrm{SAFE}^{Q,P} = \bigwedge_{w} \mathrm{SAFE}^{Q,P}_w$$

Where $\mathrm{eon}(R, s)$ *is* true *if and only if* $R(s)$ *is either even or undefined, and* $\mathrm{merge}(Q, s, R)$ *is calculated as if* s *was the color of player* O.

$$ENF_v^{\mathrm{out}}[P] = A^{\mathrm{in}}[v] \wedge \bigvee_{Q \sqsubseteq P} ENF_v^{\mathrm{in}}[Q] \wedge (s \notin \mathrm{Dom}(Q) \vee \mathrm{SAFE}^{Q,P})$$

The intuition behind $\mathrm{U}_{\mathrm{O}}^{s,t}$ is that each safe positional strategy in game G'' should factor to a strategy up to reaching a vertex of color s and a bunch of strategies after the move of player O. The part $\mathrm{SAFE}_v^{Q,P}$ corresponds to checking whether given a safe strategy with enforcement $\sqsubseteq Q$ in game G' we can make it respond to a move of player O to a vertex v in a way that maintains the safety of the strategy and makes its enforcement non-worse than P.

We merge auxiliary gadgets using $\mathrm{COMBINE}^{2,n(d+1)^k}$ with input packs being the outputs of $\mathrm{U}_{\mathrm{E}}^{s,t}$ and $\mathrm{U}_{\mathrm{O}}^{s,t}$ respectively and $COND[1] = ECOLOR[s]$, $COND[2] = \neg ECOLOR[s]$ ($\mathrm{U}_{\mathrm{E}}^{s,t}$ was created to handle the case when s is the color of player E and $\mathrm{U}_{\mathrm{O}}^{s,t}$ for the other one, here we use $\mathrm{COMBINE}^{2,n(d+1)^k}$ to choose between their results).

Note that by transitivity of $\sqsubseteq$, we have that sets ENF_v^{out} produced by $\mathrm{U}_{\mathrm{E}}^{s,t}$ and $\mathrm{U}_{\mathrm{O}}^{s,t}$ are up-closed. Note also that $\mathbb{U}^{\langle s,t\rangle}$ has constant depth and can be computed in space $\mathcal{O}(\log(n) + \log(d)k)$. This is because $\mathcal{O}(\log(n))$ bits suffice to remember a constant number of vertices. What is more, as each enforcement is a partial function from the set of k colors to the set of d ranks, so to encode an enforcement $\mathcal{O}(\log(d)k)$ bits suffice, and we consider at most four enforcements simultaneously.

▶ **Lemma 24.** *Given that* $\mathbb{U}^{\langle s,t\rangle}$ *inputs a valid set of enforcements for each vertex* v *of a subgame* G' *of game* G *then it outputs a valid set of enforcements for each vertex* v *of a game* G'' *created from* G' *by addition of directed edge from every vertex of color* s *to every vertex of color* t.

A simple case analysis suffices to prove this lemma when s belongs to player O. When s belongs to player E one can separately prove soundness (again by simple case analysis) and completeness following the presented intuition. These proofs are rather long and tedious and can be found in the full version [28].

4.3.4 Conditional Update Gadget

For each pair of colors $\langle s, t\rangle$ we define $\mathbb{U}_{cond}^{\langle s,t\rangle}$ that inputs the bits that $\mathbb{U}^{\langle s,t\rangle}$ inputs with one additional bit that is true if and only if the update defined by $\mathbb{U}^{\langle s,t\rangle}$ should be performed (i.e. whether enforcements should be updated or copied). We define $\mathbb{U}_{cond}^{\langle s,t\rangle}$ to be a simple combination of $\mathbb{U}^{\langle s,t\rangle}$ and $\mathrm{COMBINE}^{2,n(d+1)^k}$.

4.3.5 Processing Gadget

Here we define a gadget of depth $\mathcal{O}(k^2)$ that stacks operations of conditional update gadgets.

Let $p_1, \ldots, p_{k^2}$ be pairs of colors in some fixed order. We define the gadget $\mathbb{P}$ as a sequence of gadgets $\mathbb{U}_{cond}^{p_1}, \ldots \mathbb{U}_{cond}^{p_{k^2}}$ where ENF_v^{in} part of the input for gadget $\mathbb{U}_{cond}^{p_i}$ comes from the output of $\mathbb{U}_{cond}^{p_{i-1}}$ for $i > 1$ and rest of the input of $\mathbb{U}_{cond}^{p_j}$ for $j > 0$ comes from the input of $\mathbb{P}$. In particular for $\mathbb{U}_{cond}^{p_j}$ the bit that tells whether to perform the update comes from the pack of k^2 bits (denoted as DB^{in}) that $\mathbb{P}$ inputs aside from $ECOLOR$ and $COLOR_v$, ENF_v^{in}, $A^{\mathrm{in}}[v]$ for each v. We define the output of $\mathbb{P}$ to be the output of $\mathbb{U}_{cond}^{p_{k^2}}$ and $A^{\mathrm{out}}[v] = A^{\mathrm{in}}[v]$ for each v.

Note that $\mathbb{P}$ defined above has depth $\mathcal{O}(k^2)$ and can be computed in space $\mathcal{O}(\log(n) + \log(d)k)$, as each of the conditional update gadgets can be computed in this space, and we can use $\mathcal{O}(\log(k))$ bits to encode for which $\mathbb{U}^{p_j}_{cond}$ we are performing the computation.

Let l be an internal node of T from the given tree-model and let $l_1, \ldots, l_m$ be children of l. Gadget $\mathbb{P}$ will be later used to calculate valid enforcement sets for the subgame of G played on an arena induced by $\mathcal{TM}_l$ from valid enforcement sets for the subgame of G played on an union of arenas induced by $\mathcal{TM}_{l_1}, \ldots, \mathcal{TM}_{l_m}$.

4.4 Circuit Construction

The construction of the circuit will proceed in layers, with each layer output being n big packs of bits, each containing ENF^{out}_v and $A^{\text{out}}[v]$ for each vertex v, so $n(n((d+1)^k+1))$ bits in total. The output of each layer will be treated as the input for the next one. The output of l'th layer will be denoted as $LPACK_l$. To start the construction off and create layer 0, we use INIT_v gadget for each vertex v. To create parts for layer $1 \leq l \leq k$ we need to use $\mathbb{P}$ for each node $\langle l, i\rangle$. We do so by setting $\mathbb{P}$'s DB^{in} bits to $DB_{l,i}$, whereas ENF^{in}_v and $A^{\text{in}}[v]$ bits using $\text{COMBINE}^{n,n((d+1)^k+1)}$ with input being n big bit packs from $LPACK_{l-1}$ and $COND[j] = CHILD_{l,i}[j]$ (here $\text{COMBINE}^{n,n((d+1)^k+1)}$ is used to take a disjoint union of arenas). The output of $\mathbb{P}$ here constitutes to $LPACK_l$.

To produce partition of V into winning regions we first gather calculated results using $\text{COMBINE}^{n,n((d+1)^k+1)}$ gadget with input being set to n big bit packs from $LPACK_k$ and $COND[j] = \text{true}$, and then create for each vertex v gadget that checks whether there is an enforcement P such that $ENF^{\text{in}}_v[P]$ is true and for each $c \in \text{Dom}(P)$ $ECOLOR[c]$ is false.

▶ **Theorem 25.** *There exists a Turing machine that given $n, d > 1$, and $k > 0$ computes in space $\mathcal{O}(\log(n) + \log(d)k)$ circuit $\mathbb{SCW}_{\langle n,d,k\rangle}$ of depth $\mathcal{O}(k^3)$, that inputs parity game $G = (V = V_{\text{E}} \uplus V_{\text{O}}, E, rank, Color)$ and tree-model $\mathcal{TM} = (T, C = C_{\text{E}} \uplus C_{\text{O}}, Color, D)$ such that $rank(V) \subseteq \{0, \ldots, d-1\}$, $d \leq 2n$, $|C| = k$, T has height exactly k and $\mathcal{TM}$ induces G's arena, and outputs the partition of G's vertices into winning regions of players* E *and* O*.*

Proof. The complexity follows from the observations made during the construction. Correctness follows from Lemma 24 and simple induction on the size of the circuit. ◀

5 Circuit Family for Tree-Depth

Now we will show how for $n, d > 1$, and $k \geq 0$ compute in space $\mathcal{O}(\log(n) + \log(d)k)$ circuit $\mathbb{TD}_{\langle n,d,k\rangle}$ of depth being at most $\mathcal{O}(k^3)$, that given an n-vertex parity game with all ranks being less that d (we also require $d \leq 2n$) and elimination forest of height at most k, outputs the winner for each vertex. We achieve this by creating circuit $\mathbb{RED}_{\langle n,d,k\rangle}$ that changes input representation to the one accepted by $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$, and then getting the winners from the output of $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$ using circuit $\mathbb{GET}_{\langle n,d,k\rangle}$. Note that along with the construction of the circuit $\mathbb{RED}_{\langle n,d,k\rangle}$ we show a more efficient reduction that does not have an exponential blow-up in k (in contrast to the simple one presented in Section 2.4). Such improvement is not necessary for the main result of this paper, but it makes the construction more computationally tractable. In the end, we show how to get rid of the requirement for the elimination forest.

5.1 Circuit Input

Circuit $\mathbb{RED}_{\langle n,d,k\rangle}$ inputs a game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V, rank : V \to \mathbb{N})$ and elimination forest $\mathcal{F}$ as the following sequence of bits.

- For each vertex $v \in V$:
  - d bits that one-hot encode $rank(v)$ (denoted as $RANK_v$),
  - one bit that is true if and only if v is a vertex of player E (denoted as $EB[v]$),
  - $k{+}1$ bits that one-hot encode the depth of v in $\mathcal{F}$ (denoted as $DEPTH_v$; bit $DEPTH_v[x]$ is set to true if v has depth x; root has depth 0),
  - $n-1$ bits that encode the parent of v in $\mathcal{F}$ (denoted as $PARENT_v$; $PARENT_v[w]$ is true if w is a parent of v in $\mathcal{F}$; each non-root vertex has exactly one parent).
- For each pair of vertices $v, w \in V$, a bit that is set to true if and only if there is a directed edge from v to w (denoted as $EDG[v, w]$).

Checking whether $\mathcal{F}$ is a valid forest can be done similarly as in Section 4.2. For ancestor-descendant relationship one can compute in space $\mathcal{O}(\log(n) + \log(k))$ a gadget of depth $\mathcal{O}(\log(k))$ that at level i checks for every $v, w \in V$ whether there is a simple path from v to w that goes towards the root and has length at most 2^i.

5.2 Reduction Circuit

First of all, we do not aim to create exactly the same game as input for $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$, but a modified game from which we could deduce winning regions in the original one. The main idea here is to split each vertex in a number of vertices such that when we merge them back, then we get the original arena. Let $V = \{v_1, \ldots, v_n\}$ be all vertices of G, with each vertex $v \in V$ we associate $2n$ copies of v with the same player ownership: $v \to v^{\circ}_{v_1}, \ldots, v^{\circ}_{v_n}, v^{\rightarrow}_{v_1}, \ldots, v^{\rightarrow}_{v_n}$. The first n of these copies are denoted as v°, whereas the last n as $v^{\rightarrow}$. Each of them corresponds to one additional vertex from the input of $\mathbb{RED}_{\langle n,d,k\rangle}$ (the copy among the first n copies of v that corresponds to w is denoted as v°_w, whereas the copy among the last n copies that corresponds to w is denoted as $v^{\rightarrow}_w$). Intuitively for a vertex $v \in V$, copies from v° and $v^{\rightarrow}$ will be used to distribute the original connections from and to v. More precisely, vertices from v° will allow to choose a connection from v, whereas vertices from $v^{\rightarrow}$ to execute it, and if there will be a connection from v to w in G, then we will create a connection from some $v^{\rightarrow}_{v'}$ to some $w^{\circ}_{w'}$.

We set the rank of copies from $v^{\rightarrow}$ to $rank(v) + 2$, whereas the rank of copies from v° to 1 if v belongs to player E, and to 0 if v belongs to player O (as v° will be used to choose a connection we don't want a player to put off the decision indefinitely). Let x be the depth of v in $\mathcal{F}$. We color copies v° using color $\langle \circ, EB[v], x\rangle$, whereas copies from $v^{\rightarrow}$ using $\langle \rightarrow, EB[v], x\rangle$. The copies created here are leaves of the tree-model that $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$ will take as input. Note that for all operations presented above we can compute in space $\mathcal{O}(\log(n) + \log(k))$ gadget of constant depth, as we need $\mathcal{O}(\log(n))$ bits to represent a copy of a vertex and rank and $\mathcal{O}(\log(k))$ to represent a color.

Before we go any further, we introduce some gadgets. For simplicity, we will assume that vertex is its own ancestor.

- $\mathrm{CHILDLESS}_v$ – that is true if and only if v has no children in $\mathcal{F}$.
- ANEQ^w_v – that is true if and only if w is an ancestor of v in $\mathcal{F}$. Note that it can be done by a gadget with $\mathcal{O}(\log(k))$ layers computable in space $\mathcal{O}(\log(n) + \log(k))$ that in layer i considers paths from vertices towards root of length at most 2^i.
- $\mathrm{ANEQC}^{x \to x'}_v$ – that is true if and only there are two ancestor w and w' of v of depth x and x' respectively such that there is an an edge from w to w'.

Now we will describe how we compute the part of $\mathbb{RED}_{\langle n,d,k\rangle}$ that outputs the rest of the tree-model. The main thing that we will do here is that for each vertex v that is a leaf in the given elimination forest we will gather copies of ancestors that correspond to v, add connections using the copies and join the copies along the way creating a tree-model similar in shape to the given elimination forest. The computation will consist of $k+1$ major steps, and $3k+3$ simple steps. Major steps will create layers of the tree-model (see Section 4.2; layers are numbered starting from the layer of leaves, which has number 0), whereas simple steps will extend the height of the tree-model to match the specification for $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$. As in the description of the tree-model for $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$ we use graph where number of nodes in every layer is the same and in our case equals number of copies of vertices, so we will use copies of vertices to number nodes from a specific layer. The tree-model that we will produce here may not use all of the leaves (there may be vertices at the bottom that will be left isolated), we get around it by relaxing the correctness of $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$ input so that it will allow vertices from layer 0 without parents (note that they will be ignored in the further parts of the circuit). As the steps presented below are rather technical we encourage to refer to Figure 4.

First the major steps, for $l=1$, for each vertex v of the input game that is childless in $\mathcal{F}$, we gather and initialize copies of ancestors of v corresponding to v. That is for each $v,w\in V$, $p\in\{\text{true},\text{false}\}$ and $x\in\{0,\dots,k\}$ we set

$$\begin{aligned} \mathit{CHILD}_{l,v^\circ_v}[w^\circ_v] &= \mathrm{ANEQ}^w_v \wedge \mathrm{CHILDLESS}_v \\ \mathit{CHILD}_{l,v^\circ_v}[w^{\rightarrow}_v] &= \mathrm{ANEQ}^w_v \wedge \mathrm{CHILDLESS}_v \\ \mathit{DB}_{l,v^\circ_v}[\langle\langle\circ,p,x\rangle,\langle\rightarrow,p,x\rangle\rangle] &= \mathrm{CHILDLESS}_v \end{aligned}$$

Then we add all connections between gathered copies, that is for $v\in V$, $p,p'\in\{\text{true},\text{false}\}$ and $x,x'\in\{0,\dots,k\}$ we set

$$\mathit{DB}_{l,v^\circ_v}[\langle\langle\rightarrow,p,x\rangle,\langle\circ,p',x'\rangle\rangle] = \mathrm{CHILDLESS}_v \wedge \mathrm{ANEQC}^{x\to x'}_v$$

Now let $1<l\leq k+1$. Here we create parts that drag low depth vertices up in the tree-model and merge gathered copies. That is for $p,p'\in\{\text{true},\text{false}\}$, $x\in\{0,\dots,k-l+1\}$, $w\neq v$ we set:

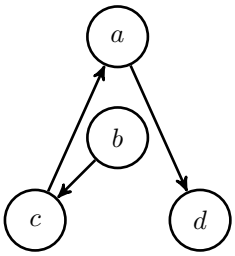
$$\begin{aligned} \mathit{CHILD}_{l,v^\circ_v}[v^\circ_v] &= \mathrm{CHILDLESS}_v \wedge \bigvee_{t\in\{0,\dots,k-l+1\}} \mathit{DEPTH}_v[t] \\ \mathit{CHILD}_{l,v^\circ_v}[w^\circ_w] &= \mathit{DEPTH}_v[k-l+1]\wedge\mathit{PARENT}_w[v] \\ \mathit{DB}_{l,v^\circ_v}[\langle\langle\circ,p,k-l+1\rangle,\langle\circ,p,k-l+1\rangle\rangle] &= \mathit{DEPTH}_v[k-l+1] \end{aligned}$$

For simple steps, we extend the height of the tree-model to match specification for $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$. Let v be some fixed vertex in V and w any vertex from V, and let $k+3\leq l\leq 4k+4$, we set $\mathit{CHILD}_{k+2,v^\circ_v}[w^\circ_w]$ to $\mathit{DEPTH}_w[0]$ whereas $\mathit{CHILD}_{l,v^\circ_v}[v^\circ_v]$ to true.

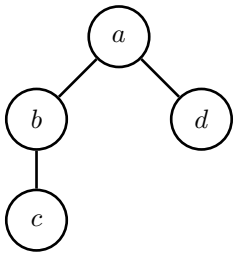
In both major and simple steps, we set unmentioned bits to false.

▶ **Proposition 26.** *The depth of $\mathbb{RED}_{\langle n,d,k\rangle}$ when constructed as above is $\mathcal{O}(\log(k))$, what is more the construction can be done in space $\mathcal{O}(\log(n)+\log(k))$.*

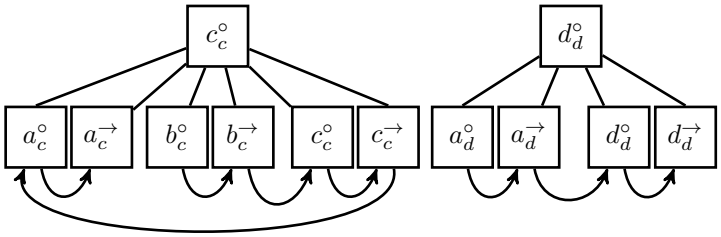
Proof. Observe that $\mathit{CHILD}_{l,v^\circ_v}$ and $\mathit{DB}_{l,v^\circ_v}$ bits depend only on bits describing depth and ancestors. What is more, $\mathcal{O}(\log(n)+\log(k))$ bits suffice to describe which step l we are performing and for which bit of either $\mathit{CHILD}_{l,v^\circ_v}$ or $\mathit{DB}_{l,v^\circ_v}$ we are creating a gate. ◀



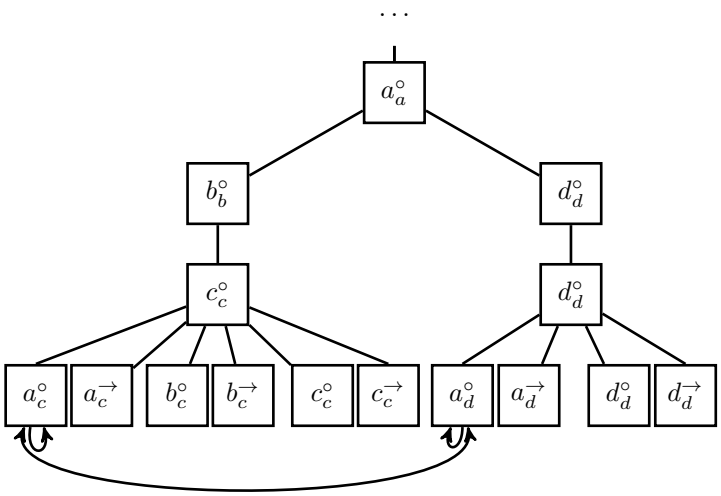
(a) Arena.



(b) Elimination forest.



(c) Layers 0 and 1 from tree-model and connections added at major step 1 (bottom).



(d) Layers $0, 1, 2, 3$ from tree-model and merge performed at major step 3 (bottom).

Figure 4 Tree-model produced for a simple arena (a) with elimination forest (b).

5.3 Output Circuit

Now we will proceed with the definition of $\mathbb{GET}_{\langle n,d,k\rangle}$. To get the winning region of E we need to check for each vertex $v \in V_{\mathrm{E}}$ whether there is a copy v_w° such that E wins from v_w° and for each $v \in V_{\mathrm{O}}$ whether E wins from all copies of v. It suffices as note that we have constructed input for $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$ such that when we contract all v° and $v^\rightarrow$ into one vertex of rank $rank(v)$ then we get the original game.

$$OUT^{\mathbb{GET}}[v] = (EB[v] \wedge \bigvee_{w \in V} OUT[v_w^\circ]) \vee (\neg EB[v] \wedge \bigwedge_{w \in V} OUT[v_w^\circ])$$

where OUT denotes the output bits of $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$. Note that $\mathbb{GET}_{\langle n,d,k\rangle}$ can be computed in space $\mathcal{O}(\log(n))$ as $\mathcal{O}(\log(n))$ bits suffice to handle a constant number of vertices.

▶ **Theorem 27.** *There exists a Turing machine that given $n, d > 1$, and $k \geq 0$ computes in space $\mathcal{O}(\log(n) + \log(d)k)$ circuit $\mathbb{TD}_{\langle n,d,k\rangle}$ of depth $\mathcal{O}(k^3)$, that inputs n-vertex parity game G along with elimination forest of G's arena, and outputs the partition of G's vertices into winning regions of players* E *and* O*.*

Proof. Follows from Proposition 26 and Theorem 25. ◀

5.4 Computing Elimination Forest

The following is an easy consequence of the result from [11, Theorem 1].

▶ **Lemma 28.** *For any $k \geq 0$ there exists a logspace uniform family of AC^0-circuits that inputs a graph and checks whether its tree-depth is k.*

Now observe that we can get an elimination forest for an arena of tree-depth at most k by a circuit that forgets about the orientation of edges and performs k steps. At step $i \in \{1, \ldots, k\}$ it considers a graph of tree-depth at most $k - i + 1$ and checks which vertices should be removed to make it a graph of tree-depth at most $k - i$. It does so by taking for each vertex v its connected component, removing v from it and checking the tree-depth of the component with v removed using a circuit from Lemma 28. In case there are many candidates the circuit picks the first one. From Lemma 28 and Lemma 5, for fixed k this circuit can have constant depth and be computed in logarithmic space (to compute connected components it suffices to check for paths of length at most $2^{k+1} - 2$ what can be done similarly as in ANEQ^w_v). So from the construction above and Theorem 27 we have the following.

▶ **Corollary 29.** *For any $k \geq 0$ there exists a logspace uniform family of AC^0-circuits that solves parity games played on arenas of tree-depth at most k.*

6 Conclusions and Further Work

We have shown that solving parity games played on arenas of bounded tree-depth or shallow clique-width is in logspace uniform AC^0, assuming that the suitable tree-model is provided for the latter case. Speaking of uniformity, one can assign appropriate binary identifiers to gates such that for any k, the problem: "Given circuit parameters n, d in binary and binary identifiers of two gates A, B decide whether in $\mathbb{TD}_{\langle n,d,k\rangle}$ both A and B are present, and the output of B constitutes to the input of A" can be solved by a deterministic Turing machine that works in time polylogarithmic in n and uses space logarithmic in n. However, going lower seems to require more technical care. A natural next step in research would be to decide whether one can obtain a similar result for the bounded shrub-depth case. However, as we observed in Section 4.1, such step seems to require a different approach.

References

1 Miklós Ajtai. $\sum^1_1$-formulae on finite structures. *Ann. Pure Appl. Log.*, 24(1):1–48, 1983. doi:10.1016/0168-0072(83)90038-6.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

3 Dietmar Berwanger, Anuj Dawar, Paul Hunter, and Stephan Kreutzer. Dag-width and parity games. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 524–536. Springer, 2006. doi:10.1007/11672142_43.

4 Dietmar Berwanger and Erich Grädel. Entanglement – A measure for the complexity of directed graphs with applications to logic and games. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Montevideo, Uruguay, March 14-18, 2005, Proceedings*, volume 3452 of *Lecture Notes in Computer Science*, pages 209–223. Springer, 2004. doi:10.1007/978-3-540-32275-7_15.

5 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263. ACM, 2017. doi:10.1145/3055399.3055409.

6 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993. doi:10.1016/0022-0000(93)90004-G.

7 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

8 Anuj Dawar and Erich Grädel. The descriptive complexity of parity games. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, volume 5213 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2008. doi:10.1007/978-3-540-87531-4_26.

9 Reinhard Diestel. *Graph Theory*. Springer Publishing Company, Incorporated, 5th edition, 2017. doi:10.1007/978-3-662-53622-3.

10 Michael Elberfeld, Martin Grohe, and Till Tantau. Where first-order and monadic second-order logic coincide. *ACM Trans. Comput. Log.*, 17(4):25, 2016. doi:10.1145/2946799.

11 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th – March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 66–77. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.66.

12 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185392.

13 John Fearnley and Sven Schewe. Time and parallelizability results for parity games with bounded treewidth. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 189–200. Springer, 2012. doi:10.1007/978-3-642-31585-5_20.

14 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.

15 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Z. Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation – 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8_15.

16 Moses Ganardi. Parity games of bounded tree- and clique-width. In Andrew M. Pitts, editor, *Foundations of Software Science and Computation Structures – 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 390–404. Springer, 2015. doi:10.1007/978-3-662-46678-0_25.

17 Robert Ganian, Petr Hlinený, Joachim Kneis, Alexander Langer, Jan Obdrzálek, and Peter Rossmanith. Digraph width measures in parameterized algorithmics. *Discret. Appl. Math.*, 168:88–107, 2014. doi:10.1016/j.dam.2013.10.038.

18 Robert Ganian, Petr Hlinený, Jaroslav Nesetril, Jan Obdrzálek, and Patrice Ossona de Mendez. Shrub-depth: Capturing height of dense graphs. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:7)2019.

19 Robert Ganian, Petr Hlinený, Jaroslav Nesetril, Jan Obdrzálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast MSO1. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 – 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2012. doi:10.1007/978-3-642-32589-2_38.

20 Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theor. Comput. Sci.*, 399(3):206–219, 2008. doi:10.1016/j.tcs.2008.02.038.

21 Marcin Jurdzinski and Ranko Lazic. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–9. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005092.

22 Karoliina Lehtinen, Pawel Parys, Sven Schewe, and Dominik Wojtczak. A recursive approach to solving parity games in quasipolynomial time. *Log. Methods Comput. Sci.*, 18(1), 2022. doi:10.46298/lmcs-18(1:8)2022.

23 Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. Practical synthesis of reactive systems from ltl specifications via parity games. *Acta Informatica*, 57(1–2):3–36, November 2019. doi:10.1007/s00236-019-00349-3.

24 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

25 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

26 Jan Obdrzálek. Fast mu-calculus model checking when tree-width is bounded. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 80–92. Springer, 2003. doi:10.1007/978-3-540-45069-6_7.

27 Jan Obdrzálek. Clique-width and parity games. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2007. doi:10.1007/978-3-540-74915-8_8.

28 Konrad Staniszewski. Parity games of bounded tree-depth, 2022. doi:10.48550/arXiv.2211.02926.