Parity Games of Bounded Tree- and Clique-Width

Moses Ganardi

University of Siegen, Germany
ganardi@eti.uni-siegen.de

Abstract. In this paper it is shown that deciding the winner of a parity game is in LogCFL, if the underlying graph has bounded tree-width, and in LogDCFL, if the tree-width is at most 2. It is also proven that parity games of bounded clique-width can be solved in LogCFL via a log-space reduction to the bounded tree-width case, assuming that a k-expression for the parity game is part of the input.

1 Introduction

Parity games are two-player graph games of infinite duration. The central question in the study of parity games is to determine the winner of a given game. This problem is motivated by its close connection to the μ-calculus model-checking problem [1] but also from a complexity theoretical perspective the problem has an interesting status: The best known upper bound is NP $\cap$ coNP (to be precise, UP $\cap$ coUP [2]) and no polynomial time algorithm is known.

In this paper we study the parallel complexity of parity games of bounded tree- and clique-width. It was shown by Obdržálek that on such classes parity games become polynomial time solvable [3,4]. Recently, Fearnley and Schewe presented an efficient parallel algorithm for parity games of bounded tree-width; more precisely, they proved that the problem belongs to NC^2 [5].

We improve the complexity bounds for parity games of bounded tree- and clique-width to LogCFL, a subclass of NC^2 containing those languages which are log-space reducible to a context-free language [6]. In the tree-width case the LogCFL bound follows from the observation that the polynomial time algorithm by Obdržálek can be simulated by a bottom-up tree automaton reading the tree decomposition. For the sake of completeness we present a new proof inspired by [7], in which hierarchically defined parity games are treated. For parity games of tree-width ≤ 2 we can improve the bound further to LogDCFL, containing those languages which are log-space reducible to a deterministic context-free language. Graphs of tree-width ≤ 2 are also known as series-parallel graphs. Finally, we prove that parity games of bounded clique-width can be log-space reduced to parity games of bounded tree-width if we assume that a k-expression for the input game is given. This yields an alternative proof for Obdržálek's clique-width result with an improved complexity bound.

2 Preliminaries

For $k \in \mathbb{N}$ we abbreviate $\{1, \ldots, k\}$ by $[k]$. For a function f we write $\mathrm{dom}(f)$ for the domain of f. We denote by $[a \mapsto b]$ the function which maps a to b, and by $f[a \mapsto b]$ the function which maps a to b and is otherwise defined as f. All graphs considered in this paper are directed graphs. We assume familiarity with the basic concepts of log-space reductions, in particular the fact that the composition of two log-space computable functions is log-space computable again. We refer to [8] for more details on parallel complexity theory.

2.1 Parity games

A *parity game* $\mathcal{G} = (V_0, V_1, E, \lambda)$ consists of a directed graph (V, E) where $V = V_0 \cup V_1$ is partitioned into vertices, or *positions*, of Player 0 and 1, and a *priority function* $\lambda : V \to \mathbb{N}$. We only consider finite parity games and define the *size* of $\mathcal{G}$ is the number of positions $|V|$. The two players move a token from a starting position along the edges forming a path π, also called *play*: if the token is currently in position $v \in V_s$ then Player s moves the token to a successor position of v. If a position $v \in V_s$ without successors is reached, then Player $1-s$ wins the play. Otherwise the play $\pi = v_0 v_1 \ldots$ is infinite and is won by Player 0 if and only if the maximal priority occurring infinitely often in $\lambda(v_0)\lambda(v_1)\ldots$ is even.

A *strategy* σ for Player s is a partial function $\sigma : V^* V_s \to V$ which maps a finite sequence $v_0 \ldots v_n$ to a successor of v_n. A play $\pi = v_0 v_1 \ldots$ is *conform* with σ if $\sigma(v_0 \ldots v_i) = v_{i+1}$ for all $i < |\pi|$ where $v_i \in V_s$. A strategy σ for Player s is *winning* from $v_0 \in V$ if Player s wins every play which is conform with σ; we also say that Player s *wins* $\mathcal{G}$ from v_0. The *winning region* of Player s is the set of all positions from which Player s wins $\mathcal{G}$. A *positional strategy* σ for Player s depends only on the current position and can be represented by a partial function $\sigma : V_s \to V$ where $(v, \sigma(v)) \in E$ for all $v \in \mathrm{dom}(\sigma)$. It is known that every parity game $\mathcal{G}$ is *positionally determined*, i.e. from every position either Player 0 or 1 has a positional winning strategy [9]. Solving parity games is formulated as the decision problem: Given a parity game $\mathcal{G}$ and a starting position $v_0 \in V$, does Player 0 win $\mathcal{G}$ from v_0?

An important parameter of a parity game $\mathcal{G}$ is the maximal priority d occurring in $\mathcal{G}$ because the running times of many algorithms for solving parity games are polynomial in the size of $\mathcal{G}$ but exponential in d. It is also known that the winning regions of parity games where the maximal priority is bounded can be defined by a fixed MSO-formula [10]. Hence, by the log-space version of Courcelle's Theorem [11] parity games whose tree-width and maximal priority are bounded by constants can be solved in log-space. For our purposes it is important that the maximal priority of a parity game can be assumed to be linear in the number of vertices by a *compression* of the priority function, see [12].

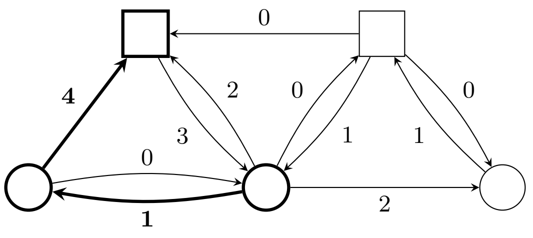


Fig. 1: An edge-labeled parity game where circles belong to Player 0 and squares belong to Player 1.

2.2 Edge-labeled parity games

For the tree-width result it is useful to convert the given parity game into a different form where priorities are assigned to edges instead of vertices. Further we allow multiple (finitely many) edges between two vertices, which is convenient for gluing together two parity games. Formally, an *edge-labeled parity game* has the form $\mathcal{G} = (V_0, V_1, E)$ where $E \subseteq V \times \mathbb{N} \times V$ is a finite set of labeled edges. In this context, a play $\pi = (v_0, p_0, v_1)(v_1, p_1, v_2)\dots$ is a finite or infinite sequence of edges, which is won by Player 0 if and only if the maximal priority occurring infinitely often in $p_0 p_1 \dots$ is even. A strategy for Player s is a partial function $\rho : V^* V_s \to E$ which maps a finite sequence $v_0 \dots v_n$ to an outgoing edge (v_n, p, v_{n+1}) of v_n. A positional strategy is a partial function $\rho : V_s \to E$, where $\rho(v)$ is an outgoing edge of v for all $v \in \mathrm{dom}(\rho)$. Winning strategies and winning regions are defined similarly as for standard parity games. Figure 1 depicts an edge-labeled parity game where the marked positions and edges form the winning region and a positional winning strategy of Player 0.

Lemma 1 *For every parity game there is an edge-labeled parity game with the same winning regions, and vice versa. In particular, edge-labeled parity games are positionally determined.*

Proof. Given a parity game, assign to each edge the priority of its starting vertex. Conversely, given an edge-labeled parity game, subdivide every edge and assign its priority to its new vertex; all other vertices have priority 0. $\square$

We will mainly deal with edge-labeled parity games $\mathcal{G}$ without multiple edges, i.e. $(u, p, v), (u, q, v) \in E$ implies $p = q$, which we call *simple (edge-labeled) parity games.* Whenever an edge-labeled parity game contains exactly one edge between a pair of positions u and v, we sometimes denote the edge by (u, v) without the priority and we write $\lambda(u, v) = p$. Every edge-labeled parity game can be made simple as witnessed by the following lemma.

Lemma 2 *For every edge-labeled parity game one can compute in log-space a simple edge-labeled parity game with the same winning regions.*

Proof. Consider the *reward order* $\sqsubseteq$ on $\mathbb{N}$, which intuitively sorts the priorities according to their attractivity to Player 0: We define $p \sqsubseteq q$ if p and q are even and $p \leq q$, or p and q are odd and $p \geq q$, or p is odd and q is even.

For an edge-labeled parity game $\mathcal{G} = (V_0, V_1, E)$ we define the *simplified* edge-labeled parity game $\mathsf{simple}(\mathcal{G})$ by combining multiple edges between two endpoints (u, v) into a single one with the following priority:

$$\lambda(u,v) = \begin{cases} \max_{\sqsubseteq}\{p \in \mathbb{N} : (u,p,v) \in E\}, & \text{if } u \in V_0, \\ \min_{\sqsubseteq}\{p \in \mathbb{N} : (u,p,v) \in E\}, & \text{if } u \in V_1. \end{cases}$$

It can be verified that winning regions are preserved. □

2.3 Tree-width

In the following we define two well-known graph decompositions and the corresponding graph measures, tree-width and clique-width. Many NP-complete problems become solvable in log-space or linear time on classes of bounded tree- or clique-width, see [11,13,14].

A *tree decomposition* $\mathcal{T} = (T, \{X_i\}_{i \in I})$ of a graph $G = (V, E)$ consists of a rooted tree T with node set I and a family of *bags* $X_i \subseteq V$ for $i \in I$ such that for all $(u, v) \in E$ there exists $i \in I$ with $u, v \in X_i$, and for all $v \in V$ the set $\{i \in I : v \in X_i\}$ is non-empty and connected in T. The *width* of $\mathcal{T}$ is $\max_{i \in I} |X_i| - 1$ and the *tree-width* of a graph G is the minimum width of a tree decomposition of G. The tree-width of a parity game is the tree-width of its underlying graph. Deciding whether the tree-width of a given graph is at most a given parameter $k \in \mathbb{N}$ is NP-complete [15]; however, for every fixed $k \in \mathbb{N}$ there exists a log-space algorithm which decides whether a given graph has tree-width $\leq k$ and in that case computes a width-k tree decomposition for it [11]. We call a width-k tree decomposition $\mathcal{T} = (T, \{X_i\}_{i \in I})$ *smooth* if

(a) $|X_i| = k + 1$ for all $i \in I$,
(b) $|X_i \cap X_j| = k$ for all edges (i, j) in T.

It is known that tree decompositions can be made smooth in linear time, see [16, Chapter 6]. For our purposes we devise a space efficient algorithm:

Lemma 3 *For every fixed $k \in \mathbb{N}$ there exists a log-space algorithm which, given a width-k tree decomposition $\mathcal{T}$ of G, computes a smooth width-k tree decomposition of G.*

Proof. Let $\mathcal{T} = (T, \{X_i\}_{i \in I})$ be a tree decomposition of width k. First of all, we root T at some node $i \in I$ such that $|X_i| = k + 1$, which is computable in log-space.

For property (a) we present a procedure which adds a vertex to each bag X_i which does not have maximal size. Let $I_0 = \{i \in I : |X_i| = k + 1\}$. For each $i \in I \setminus I_0$ independently we add a vertex to X_i as follows: Let j be the lowest ancestor of i in I_0 and let j' be the unique child of j on the path from j to i.

We add the lexicographically smallest vertex in $X_j \setminus X_{j'}$ to X_i. This procedure preserves the tree decomposition properties and can be performed in log-space. After at most $k+1$ iterations of this procedure, all bags have uniform size.

For property (b) consider an inclusion maximal set $J \subseteq I$ where $X_i = X_j$ for all $i, j \in J$, which forms a connected subtree of T. All such sets J can be found in log-space and can be merged into single nodes such that all bags are pairwise distinct. Now assume $|X_i \cap X_j| < k$ for some tree edge (i,j). Let $X_i \setminus X_j = \{u_1, \dots, u_m\}$ and $X_j \setminus X_i = \{v_1, \dots, v_m\}$. We replace the edge (i,j) by a path $(i_0, \dots, i_m)$ where $X_{i_\ell} = (X_i \cap X_j) \cup \{v_1, \dots, v_\ell, u_{\ell+1}, \dots, u_m\}$. $\square$

2.4 Clique-width

To define clique-width we need to consider *colored graphs* $G = (V, E, \gamma)$ where $\gamma : V \to [k]$ is a *coloring function.* A *k-expression* is a term built up from constants $i \in [k]$, unary symbols ρ_β and $\alpha_{i,j}$ where $\beta : [k] \to [k]$ and $i, j \in [k]$, and a binary symbol $\oplus$. Every k-expression t defines a colored graph $\mathrm{val}(t)$ up to isomorphism as follows:

– $\mathrm{val}(i) = (\{v\}, \emptyset, [v \mapsto i])$ where v is a fresh symbol.
– $\mathrm{val}(t_1 \oplus t_2)$ is the disjoint union of $\mathrm{val}(t_1)$ and $\mathrm{val}(t_2)$.
– If $\mathrm{val}(t) = (V, E, \gamma)$, we set $\mathrm{val}(\rho_\beta(t)) = (V, E, \beta \circ \gamma)$.
– If $\mathrm{val}(t) = (V, E, \gamma)$, we set $\mathrm{val}(\alpha_{i,j}(t)) = (V, E', \gamma)$ where

$$E' = E \cup \{(v, w) \in V^2 : \gamma(v) = i, \gamma(w) = j\}.$$

The *clique-width* of a graph G is the minimal number $k \in \mathbb{N}$ such that there is a k-expression t and a coloring function γ of G with $\mathrm{val}(t) = (G, \gamma)$. We remark that the standard definition of k-expressions uses operations of the form $\rho_{i \to j}$ which recolors all vertices with color i to j; this does not affect the definition of clique-width. To define the clique-width of parity games we modify the form of k-expressions to define *colored parity games* $(\mathcal{G}, \gamma)$. In this context a k-expression is built up from constants $(i, s, p) \in [k] \times \{0, 1\} \times \mathbb{N}$, which defines a parity game with a single vertex of Player s with color i and priority p, and the unary and binary symbols as before.

It is known that every graph class of bounded tree-width has also bounded clique-width but not vice versa [17]. In that sense clique-width is a more general graph measure than tree-width. As with tree-width, deciding whether the clique-width of a given graph is at most a given parameter is NP-complete [18]. Unlike tree-width it is open whether for fixed $k \geq 4$ the question, does a given graph have clique-width $\leq k$, is solvable in polynomial time. In Section 5 we will assume that a k-expression for the parity game is already part of the input.

2.5 Tree automata

We consider terms (or *trees*) over a *ranked alphabet Σ*, i.e. a finite set of function symbols where every symbol has an arity. A *(bottom-up) tree automaton*

$\mathcal{A} = (Q, \Delta, F)$ over Σ consists of a finite set of *states* Q, a set Δ of *transition rules* of the form $a(q_1, \ldots, q_n) \to q$ where $a \in \Sigma$ is n-ary and $q_1, \ldots, q_n, q \in Q$, and a set of *final states* $F \subseteq Q$. We call $\mathcal{A}$ *deterministic* if there are no two rules in Δ with the same left-hand side, otherwise $\mathcal{A}$ is called *nondeterministic*. A tree t is *accepted* by $\mathcal{A}$ if $t \xrightarrow{*}_{\Delta} q$ for some $q \in F$ where $\to_\Delta$ is the *one-step rewriting relation* defined by Δ. The uniform membership problem for (non)deterministic tree automata asks: Does a given (non)deterministic tree automaton accept a given tree? It is known that the uniform membership problem is LogCFL-complete in the nondeterministic case and in LogDCFL in the deterministic case [19].

3 Parse trees

Instead of working directly with tree decompositions our algorithm for parity games of bounded tree-width uses an equivalent notion from [16, Chapter 6], called *parse trees*, which describe how a graph or a parity game of bounded tree-width can be constructed using simple operations.

3.1 Parse trees for graphs

A *k-graph* (V, E, τ) is a graph together with an injective function $\tau : [k] \to V$, which distinguishes k vertices, called *boundary vertices*. Vertices which are not boundary are called *internal*. Given k-graphs $G = (V, E, \tau)$, $G' = (V', E', \tau')$ we define the following *parsing operators*:

- $\mathsf{rename}_\beta(G) = (V, E, \tau \circ \beta^{-1})$ where β is a permutation of $[k]$,
- $\mathsf{push}(G) = (V \cup \{v\}, E, \tau[1 \mapsto v])$ where v is a fresh symbol,
- $\mathsf{glue}(G, G')$ takes the disjoint union of two k-graphs, and identifies $\tau(i)$ and $\tau'(i)$ for all $i \in [k]$.

If $\mathsf{glue}(G, G') = (V'', E'', \tau'')$, we assume for simplified notation that $V \cup V' = V''$ and $\tau = \tau' = \tau''$, i.e. a boundary vertex in $\mathsf{glue}(G, G')$ has the same name in G and in G'. We will consider graphs constructed by combining *atomic* k-graphs, which are k-graphs of size k, with the parsing operators. A *parse tree* t of width k is the representation of such a construction as a labeled tree. A parse tree t *defines* a k-graph G if G is isomorphic to the k-graph obtained by evaluating t bottom-up; we also simply say that G is *definable* if the parse tree is irrelevant. A parse tree t *defines* a graph G if t defines (G, τ) for some τ.

It was shown in [16] that a graph has tree-width $\leq k$ if and only if it is definable by a parse tree of width $\leq k+1$. Their proof shows that the conversion from tree decompositions to parse trees can be carried out in linear time. Using smooth tree decompositions we prove that parse trees can be computed in log-space for graphs of bounded tree-width.

Lemma 4 *For every fixed $k \in \mathbb{N}$, there exists a log-space algorithm which, given a graph G of tree-width $\leq k$, computes a parse tree of width $\leq k+1$ defining G.*

Proof. Let $\mathcal{T} = (T, \{X_i\}_{i \in I})$ be a width-k tree decomposition of G computed by the log-space algorithm from [11]. By Lemma 3 we can make $\mathcal{T}$ smooth in log-space. For each $i \in I$ we fix a numbering of the $k+1$ vertices in X_i. Let G_i be the $(k+1)$-graph induced by the subtree of T rooted in i where the j-th boundary vertex of G_i is the j-th vertex in X_i.

We present a bottom-up construction for the $(k+1)$-graphs G_i. If $i \in I$ is a leaf node, then G_i is atomic. If $i \in I$ is an inner node, for each child $j \in I$ of i we apply the following operations to G_j to obtain G'_j: Assume $X_j \setminus X_i = \{v\}$ and $X_i \setminus X_j = \{w\}$. Permute v to be the first boundary vertex, introduce w using push, add existing edges between w and the other boundary vertices by gluing with a suitable atomic $(k+1)$-graph and permute the boundary vertices according to the order on X_i. By gluing all such graphs G'_j we obtain a $(k+1)$-graph isomorphic to G_i. This construction gives rise to a parse tree, which can be computed in log-space from $\mathcal{T}$. $\square$

3.2 Parse trees for parity games

We want to transfer the notion of parse trees to parity games. For that we consider *k-games*, i.e. edge-labeled parity games $\mathcal{G} = (V_0, V_1, E, \tau)$ with k *boundary positions* given by an injective function $\tau : [k] \to V$. The operator rename_β is defined as previously. The operator push_s carries a parameter $s \in \{0, 1\}$ which specifies that the new position belongs to Player s. Finally $\mathsf{glue}(\mathcal{G}_1, \mathcal{G}_2)$ is only defined if $\mathcal{G}_1$ and $\mathcal{G}_2$ are *compatible*, i.e. corresponding boundary positions belong to the same player. Parse trees for k-games are defined in an analogous manner. Here the leaf nodes are labeled by *atomic k-games*, i.e. k-games of size k.

As a precomputation step of the algorithms in the next section we compute for a given parity game $\mathcal{G}$ and a starting position v_0 a parse tree t of width k which defines a k-game in which v_0 is a boundary position. This can be done in log-space by an adaption of Lemma 4: First we compute in log-space a smooth tree decomposition of the underlying graph of $\mathcal{G}$. Then we root the tree decomposition at some bag containing v_0 before converting it into a parse tree. In the conversion phase we label the leaf nodes of the parse tree by the atomic k-games induced by the leaf bags. Whenever a new position v is introduced by push, we annotate the push-operator by the parameter $s \in \{0, 1\}$ depending on the owner of v.

4 Parity games of bounded tree-width

In this section by parity games we always mean edge-labeled parity games. Consider the construction of a definable k-game $\mathcal{G}$ from atomic k-games using the parsing operators, as described by a parse tree. We reduce the problem of determining the winner of $\mathcal{G}$ from some boundary position to the evaluation of a tree automaton reading the parse tree. One possible approach is to compute in a bottom-up manner for each tree node a small k-game which is equivalent in a certain sense to the k-game defined by the subtree rooted in that node. In the end it remains to solve a small parity game in the root node. In fact, every definable 3-game has an equivalent simple atomic 3-game. Simple parity games of constant size can be stored in space $\mathcal{O}(\log d)$ where d is the maximal priority. However, this approach fails for definable k-games where $k > 3$, i.e. parity games with tree-width > 2. Instead, with the help of a nondeterministic tree automaton we guess and fix a positional strategy for Player 0, and obtain a parity game in which only Player 1 makes non-trivial moves. So called solitaire k-games can again be compressed to size k. In both approaches the tree automata can be constructed using only logarithmic space. Since the uniform membership problems for the corresponding tree automata can be solved in LogCFL and LogDCFL, respectively, parity games of bounded tree-width can be solved in LogCFL and parity games of tree-width ≤ 2 can be solved in LogDCFL.

4.1 Equivalent k-games and valid reduction rules

We start by defining the following Myhill-Nerode type equivalence: Two compatible k-games $\mathcal{G}_1, \mathcal{G}_2$ are called *equivalent*, denoted by $\mathcal{G}_1 \approx \mathcal{G}_2$, if for all k-games $\mathcal{H}$ compatible with $\mathcal{G}_1$ and $\mathcal{G}_2$ and all positions v in $\mathcal{H}$ we have

$$\text{Player 0 wins } \mathsf{glue}(\mathcal{G}_1, \mathcal{H}) \text{ from } v \iff \text{Player 0 wins } \mathsf{glue}(\mathcal{G}_2, \mathcal{H}) \text{ from } v.$$

In fact, $\approx$ is a congruence with respect to the parsing operators, i.e. $\mathcal{G}_1 \approx \mathcal{G}_2$ implies $\mathsf{rename}_\beta(\mathcal{G}_1) \approx \mathsf{rename}_\beta(\mathcal{G}_2)$, $\mathsf{push}_s(\mathcal{G}_1) \approx \mathsf{push}_s(\mathcal{G}_2)$ and $\mathsf{glue}(\mathcal{G}_1, \mathcal{H}) \approx \mathsf{glue}(\mathcal{G}_2, \mathcal{H})$ for all k-games $\mathcal{H}$ compatible with $\mathcal{G}_1$ and $\mathcal{G}_2$.

We introduce *valid reduction rules*, which compute in log-space to a given k-game $\mathcal{G}$ an equivalent k-game $\mathcal{G}'$. For example, the operation simple from Lemma 2, which removes multiple edges, is valid, which can be shown by a very similar proof. We will always append the application of simple to valid reduction rules without mentioning it. In the following let $\mathcal{G} = (V_0, V_1, E, \tau)$ be a simple k-game. For the sake of easier proofs we can assume that each player uses a *uniform* positional winning strategy, which is winning from all positions in his winning region [20, Chapter 6]. A positional strategy ρ *uses* an edge $(u, p, v) \in E$ if $\rho(u) = (u, p, v)$. One can also assume that a positional winning strategy ρ for Player s is *minimal*, i.e. $\mathrm{dom}(\rho)$ is contained in the winning region of Player s. In the following, if we mention winning strategies, we always mean uniform minimal positional winning strategies.

Lemma 5 *Let $u \in V_s$ be an internal position where $(u, p, u) \in E$.*

1. *If $p \equiv s \pmod 2$, it is valid to add loops with priority s to all predecessors of u in V_s and then to remove u.*
2. *If $p \not\equiv s \pmod 2$, it is valid to remove the loop (u, p, u).*

Proof. Let $\mathcal{H}$ be a k-game compatible with $\mathcal{G}$ and let $\mathcal{G}'$ be the modified k-game.

1. If a winning strategy for Player s in $\mathsf{glue}(\mathcal{G}, \mathcal{H})$ uses an edge leading to u, Player s can instead use the new loops with priority s in $\mathsf{glue}(\mathcal{G}', \mathcal{H})$. On the other hand no winning strategy for Player $1-s$ in $\mathsf{glue}(\mathcal{G}, \mathcal{H})$ can use one of the edges leading to u.

2. The loop (u, p, u) cannot be used by any winning strategy for Player s in $\mathsf{glue}(\mathcal{G}, \mathcal{H})$. Hence, $\mathsf{glue}(\mathcal{G}, \mathcal{H})$ and $\mathsf{glue}(\mathcal{G}', \mathcal{H})$ have the same winning regions. □

Lemma 6 *Let $u, v \in V_s$ where $(v, u) \in E$. It is valid to add edges (v, p, w) where $p = \max(\lambda(v, u), \lambda(u, w))$ for each successor w of u, and then to remove the edge (v, u).*

Proof. Let $\mathcal{H}$ be a k-game compatible with $\mathcal{G}$, let $\mathcal{G}'$ be the modified k-game and let ρ be a winning strategy for Player s in $\mathsf{glue}(\mathcal{G}, \mathcal{H})$. If $\rho(v) = (v, u)$, then $\rho(u)$ must be defined, say $\rho(u) = (u, w)$. In $\mathsf{glue}(\mathcal{G}', \mathcal{H})$ Player s can win from the same winning region by moving from v directly to w and otherwise playing according to ρ. Conversely, if ρ is a winning strategy for Player s in $\mathsf{glue}(\mathcal{G}', \mathcal{H})$ which uses one of the new edges (v, p, w), then Player s can win from the same winning region by moving from v via u to w, and otherwise playing according to ρ. □

Lemma 7 *Let $(u, p, v), (v, q, u) \in E$ where u and v belong to different players and let $s = \max(p, q) \bmod 2$. It is valid to remove the edge from the cycle whose starting point belongs to Player $1 - s$.*

Proof. Let $\mathcal{H}$ be a k-game compatible with $\mathcal{G}$. No winning strategy for Player $1 - s$ in $\mathsf{glue}(\mathcal{G}, \mathcal{H})$ can use the edge which is to be removed because Player s could win by responding with the other edge. □

Let $\mathcal{C}$ be a class of k-games and let f be an n-ary partial operation mapping n-tuples $\overline{\mathcal{G}} = (\mathcal{G}_1, \ldots, \mathcal{G}_n)$ of k-games to a k-game $f(\overline{\mathcal{G}})$. A partial operation $f' : \mathcal{C}^n \to \mathcal{C}$ *implements* f *on* $\mathcal{C}$ if $f(\overline{\mathcal{G}}) \approx f'(\overline{\mathcal{G}})$ for all $\overline{\mathcal{G}} \in \mathrm{dom}(f) \cap \mathcal{C}^n$. With the help of the previous lemmata we can prove the following main ingredient for solving parity games of tree-width at most 2 using deterministic tree automata.

Lemma 8 *All parsing operators have log-space computable implementations on the class of all simple atomic 3-games.*

Proof. On the class of all simple atomic k-games rename_β implements itself and $\mathsf{simple} \circ \mathsf{glue}$ implements glue. It remains to treat the parsing operator push_s.

If $\mathcal{G}$ is an atomic 3-game, then $\mathsf{push}_s(\mathcal{G})$ is the disjoint union of a 2-game of size 3 and a boundary position belonging to Player s. Since the addition of an isolated boundary position respects $\approx$, it suffices to show that for each simple 2-game of size 3 one can compute in log-space an equivalent 2-game of size 2.

So let $\mathcal{G} = (V_0, V_1, E, \tau)$ be a simple 2-game of size 3. Let $v_i = \tau(i)$ for $i \in \{1, 2\}$ and let $u \in V_s$ be the unique internal position. By applying the reduction rules from Lemma 5, 6 and 7 to u, we can eliminate all cycles of length at most 2 in $\mathcal{G}$ which contain u. Hence, between u and each v_i there exists at most one edge in one direction. We can eliminate u using the following valid reduction rules:

1. If u has no incoming edge, remove u.

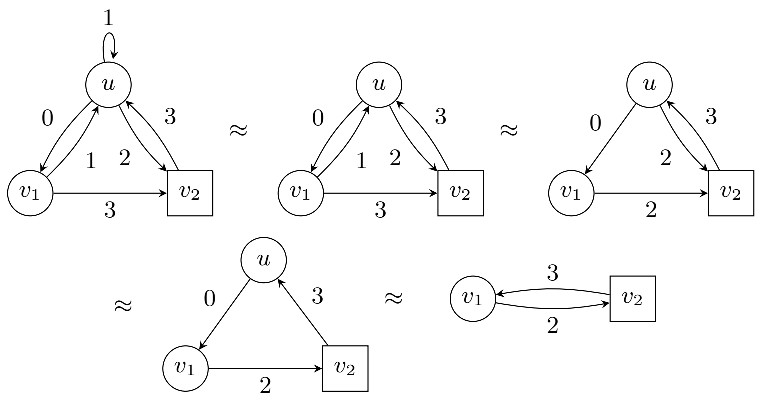


Fig. 2: Applying valid reduction rules to a 2-game.

2. If u has no outgoing edge, add loops with priority $1 - s$ to all predecessors of u in V_{1-s} and remove u.
3. Otherwise the only edges incident to u are $(v_i, u), (u, v_j) \in E$ for $i \neq j$. Add an edge (v_i, v_j) with priority $\max(\lambda(v_i, u), \lambda(u, v_j))$ and remove u. $\square$

Later we will see that Lemma 8 cannot be extended to k-games for $k > 3$. However, if we fix a positional strategy of Player 0 it suffices to consider parity games in which Player 1 makes non-trivial moves. A *solitaire game* for Player s is a parity game where all positions belong to Player s.

Lemma 9 *For every $k \in \mathbb{N}$ and $s \in \{0, 1\}$, all parsing operators except for* push_{1-s} *have log-space computable implementations on the class of all simple atomic solitaire k-games for Player s.*

Proof. As in Lemma 8 we only need to show that every simple atomic solitaire k-game $\mathcal{G}$ of size $k + 1$ can be compressed to size k in log-space. Using Lemma 5 and 6 we can eliminate all incoming edges of the unique internal position u in $\mathcal{G}$. Then u can be removed. $\square$

4.2 Construction of the tree automata

We fix the following (arbitrary) encoding of isomorphism classes of k-games. An atomic k-game $\mathcal{G} = (V_0, V_1, E, \tau)$ is in *normal form* if $V = [k]$ and $\tau(i) = i$ for all $i \in [k]$. Given an atomic k-game $\mathcal{G}$, we denote by $[\mathcal{G}]$ the unique k-game in normal form isomorphic to $\mathcal{G}$.

Theorem 10 *Parity games of tree-width ≤ 2 can be solved in LogDCFL.*

Proof. Let $\mathcal{G}_0$ be a parity game of tree-width ≤ 2 with maximal priority d and let v_0 be a given starting position. We apply simple to $\mathcal{G}_0$ and compute a parse tree t of width 3 as explained in Section 3.2 such that v_0 is the i-th boundary position of the 3-game defined by t. We assume that the atomic 3-games in the leaf nodes of t are in normal form.

Let $\mathcal{C}$ be the set of simple atomic 3-games in normal form with maximal priority $\leq d$. We can encode the elements $\mathcal{G} \in \mathcal{C}$ using $\mathcal{O}(\log d)$ bits where d was assumed to be linear in the size of $\mathcal{G}$. We compute from the parameters d and i a deterministic tree automaton $\mathcal{A} = (\mathcal{C}, \Delta, F)$ over the alphabet of t, where Δ contains for all compatible $\mathcal{G}, \mathcal{H} \in \mathcal{C}$ transitions of the form

$$\begin{aligned} \mathcal{G} &\to \mathcal{G}, \\ \mathsf{rename}_\beta(\mathcal{G}) &\to [\mathsf{rename}'_\beta(\mathcal{G})], \\ \mathsf{push}_s(\mathcal{G}) &\to [\mathsf{push}'_s(\mathcal{G})], \\ \mathsf{glue}(\mathcal{G}, \mathcal{H}) &\to [\mathsf{glue}'(\mathcal{G}, \mathcal{H})]. \end{aligned}$$

Here f' denotes the implementation of f from Lemma 8. A state $\mathcal{G}$ is contained in F if and only if Player 0 wins $\mathcal{G}$ from the i-th boundary position, which can be easily computed in log-space. Player 0 wins $\mathcal{G}_0$ from v_0 if and only if $\mathcal{A}$ accepts t, which can be decided in LogDCFL [19]. □

For our approach to solve parity games with tree-width > 2 it is convenient to assume that every position has at least one successor, which can be established by adding to each position of Player s a loop with priority $1 - s$. Let ρ be a positional strategy for Player 0 in an edge-labeled parity game $\mathcal{G} = (V_0, V_1, E)$. We define $\mathcal{G}_\rho = (\emptyset, V_0 \cup V_1, E_\rho)$ where

$$E_\rho = \{\rho(v) : v \in \mathrm{dom}(\rho)\} \cup \{(v, p, w) \in E : v \in V_1\},$$

which is a solitaire game for Player 1. It is easy to see that Player 0 wins $\mathcal{G}$ from v if and only if there exists a positional strategy ρ for Player 0 with $\mathrm{dom}(\rho) = V_0$ such that Player 0 wins $\mathcal{G}_\rho$ from v. A nondeterministic automaton reading a parse tree can guess and fix positional strategies for Player 0 on the atomic k-games in the leaf nodes and verify whether they together form a positional strategy ρ in the whole game such that $\mathrm{dom}(\rho) = V_0$.

Theorem 11 *Parity games of bounded tree-width can be solved in LogCFL.*

Proof. We adapt the proof of Theorem 10 where $\mathcal{G}_0$ now has tree-width $\leq k$. We compute in log-space a parse tree t which defines a $(k+1)$-game $(\mathcal{G}_0, \tau)$ such that $\tau(i) = v_0$. Let $M = \{j \in [k+1] : \tau(j) \in V_0\}$ and let $\mathcal{C}$ be the set of simple atomic solitaire $(k+1)$-games for Player 1 in normal form with maximal priority $\leq d$. We define $\mathcal{A} = (\mathcal{C} \times 2^{[k+1]}, \Delta, F)$ over the alphabet of t, where Δ contains for all compatible $\mathcal{G}, \mathcal{H} \in \mathcal{C}$, subsets $U, W \subseteq [k+1]$ and positional strategies ρ for Player 0 on $\mathcal{G}$, transitions of the form

$$\begin{aligned} \mathcal{G} &\to (\mathcal{G}_\rho, \mathrm{dom}(\rho)), \\ \mathsf{rename}_\beta((\mathcal{G}, U)) &\to ([\mathsf{rename}'_\beta(\mathcal{G})], \beta(U)), \\ \mathsf{push}_s((\mathcal{G}, U)) &\to ([\mathsf{push}'_1(\mathcal{G})], U \setminus \{1\}), \\ \mathsf{glue}((\mathcal{G}, U), (\mathcal{H}, W)) &\to ([\mathsf{glue}'(\mathcal{G}, \mathcal{H})], U \cup W), \quad \text{if } U \cap W = \emptyset. \end{aligned}$$

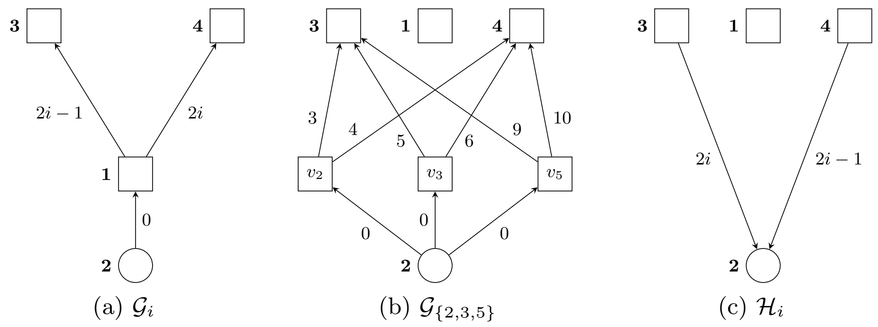


Fig. 3: 4-games for the separation of $\approx$-classes.

Here f' denotes the implementation of f from Lemma 9. A state $(\mathcal{G}, U)$ is in F if and only if Player 0 wins $\mathcal{G}$ from the i-th boundary position and $U = M$. We can encode all states using $\mathcal{O}(\log d)$ bits and compute Δ and F in log-space. Player 0 wins $\mathcal{G}_0$ from v_0 if and only if $\mathcal{A}$ accepts t, which can be decided in LogCFL. □

4.3 A lower bound

We conclude this section with a proof that parity games of tree-width 3 cannot be solved in LogDCFL using the deterministic tree automata approach as in Theorem 10.

Theorem 12 *For each $d \in \mathbb{N}$ there exist $2^d - 1$ many definable 4-games which have maximal priority $\leq 2d$ and are pairwise inequivalent.*

Proof. Consider the atomic 4-games $\mathcal{G}_i$ and $\mathcal{H}_i$ depicted in Figure 3. For every non-empty subset $I \subseteq [d]$ we construct a 4-game $\mathcal{G}_I$ by gluing all 4-games $\mathsf{push}_1(\mathcal{G}_i)$ for $i \in I$ together. In $\mathsf{glue}(\mathcal{G}_I, \mathcal{H}_i)$ we denote by u the unique position of Player 0 and by v_k the position whose outgoing edges are labeled by $2k-1$ and $2k$ for all $k \in I$.

We claim that for all $i \in [d]$ and non-empty $I \subseteq [d]$, Player 0 wins $\mathsf{glue}(\mathcal{G}_I, \mathcal{H}_i)$ from u if and only if $i \in I$, which proves that all 4-games $\mathcal{G}_I$ are pairwise inequivalent. If $i \in I$, Player 0 wins $\mathsf{glue}(\mathcal{G}_I, \mathcal{H}_i)$ by always moving from u to v_i. If $i \notin I$, Player 1 wins $\mathsf{glue}(\mathcal{G}_I, \mathcal{H}_i)$ as follows. From a position v_k Player 1 moves along the edge labeled by $2k-1$ if $k > i$, and along the edge labeled by $2k$ if $k < i$. □

For numbers $k, d \in \mathbb{N}$ and $i \in [k]$, consider the tree language of all parse trees which define k-games with maximal priority $\leq d$ won by Player 0 from the i-th boundary position. This tree language is regular by Theorem 11 but already for $k = 4$ it cannot be recognized by a deterministic tree automaton with a polynomial number of states in d according to Theorem 12. It remains open whether the presented complexity bounds for parity games of bounded tree-width can be improved.

5 Parity games of bounded clique-width

In this final section we present a log-space reduction which transforms parity games given by k-expressions into parity games of tree-width $\leq 8k-1$ and preserves the winners. As a corollary we obtain the following theorem:

Theorem 13 *For every $k \in \mathbb{N}$, parity games of clique-width $\leq k$ can be solved in LogCFL, assuming that a k-expression for the parity game is part of the input.*

Let $t_{\mathcal{G}}$ be a k-expression which defines a parity game $\mathcal{G} = (V_0, V_1, E, \lambda)$. We view $t_{\mathcal{G}}$ as a labeled tree and define T to be the set of all tree nodes, i.e. all subterms of $t_{\mathcal{G}}$. For each $v \in V$ we denote by $t_v \in T$ the unique leaf node which introduces v. Recall that t_v specifies the color of v when first introduced, which we denote by $\gamma(v)$, the owner of v and its priority $\lambda(v)$.

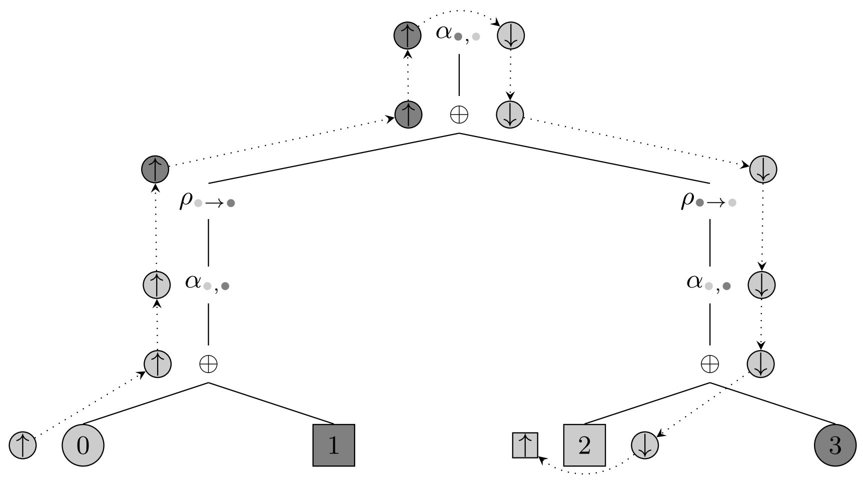
We simulate $\mathcal{G}$ by the *tree game* $\mathcal{G}^*$, which is basically played on the tree $t_{\mathcal{G}}$. During a play in the tree game we need to memorize additional information, which is why we have multiple copies of each tree node. The positions in the tree game $\mathcal{G}^*$ are of the form (t, i, m, s) where

- $t \in T$ is a tree node,
- $i \in [k]$ is the current color,
- $m \in \{\uparrow, \downarrow\}$ specifies the current direction and
- $s \in \{0, 1\}$ indicates that the position (t, i, m, s) belongs to Player s.

For every position $v \in V_s$ in $\mathcal{G}$ we define the corresponding position $v^{\uparrow} = (t_v, \gamma(v), \uparrow, s)$ in $\mathcal{G}^*$. The edges of $\mathcal{G}^*$ are defined in the following: Player s can draw from a position $(t, i, \uparrow, s)$ to $(t', j, \uparrow, s)$ where t' is the father node of t and, if $t' = \rho_\beta(t)$, then $\beta(i) = j$, otherwise $i = j$. In a position of the form $(\alpha_{i,j}(t), i, \uparrow, s)$ Player s can decide to draw to $(\alpha_{i,j}(t), j, \downarrow, s)$. Then, Player s can draw from a position $(t, i, \downarrow, s)$ to $(t', j, \downarrow, s)$ where t' is a child node of t and, if $t = \rho_\beta(t')$ then $\beta(j) = i$, otherwise $i = j$. From a position of the form $(t_w, \gamma(w), \downarrow, s)$ Player s has to draw to $w^{\uparrow}$ from where the owner of w continues to play. Note that there are positions without outgoing edges in the tree game, for example positions $(t_v, i, \downarrow, s)$ where $i \neq \gamma(v)$ or positions $(\alpha_{i,j}(t), k, \uparrow, s)$ where $i \neq k$ and $\alpha_{i,j}(t)$ is the root of $t_{\mathcal{G}}$. For all $v \in V$ we assign the priority $\lambda(v)$ to the position $v^{\uparrow}$ and to all other positions priority 0. Clearly, $\mathcal{G}^*$ can be computed in log-space from $t_{\mathcal{G}}$. The following lemma shows how plays in $\mathcal{G}$ can be simulated in the tree game.

Lemma 14 *Player 0 wins $\mathcal{G}$ from $v_0 \in V$ if and only if she wins the tree game $\mathcal{G}^*$ from $v_0^{\uparrow}$.*

Proof. A move from $v \in V_s$ to a successor w in $\mathcal{G}$ can be simulated by a finite path π_{vw} in $\mathcal{G}^*$ from $v^{\uparrow}$ to $w^{\uparrow}$. Let $\alpha_{i,j}(t)$ be a tree node that introduces the edge (v, w). Player s moves "upwards" to $(\alpha_{i,j}(t), i, \uparrow, s)$, then to $(\alpha_{i,j}(t), j, \downarrow, s)$. After that Player s moves "downwards" to $(t_w, \gamma(w), \downarrow, s)$ and finally to $w^{\uparrow}$. In this way, every positional strategy σ for Player s in $\mathcal{G}$ defines a strategy σ^* for Player s in $\mathcal{G}^*$, which in general is not positional. The maximal priority of a



$\alpha_{\bullet,\bullet}$
$\oplus$
$\rho_{\bullet\to\bullet}$
$\rho_{\bullet\to\bullet}$
$\alpha_{\bullet,\bullet}$
$\alpha_{\bullet,\bullet}$
$\oplus$
$\oplus$
0
1
2
3

Fig. 4: A 2-expression and a finite play in the tree game.

position in π_{vw} is $\max(\lambda(v), \lambda(w))$. Also notice that $v \in V_s$ has no outgoing edges in $\mathcal{G}$ if and only if no other position of the form $w^\uparrow$ is reachable from $v^\uparrow$ in $\mathcal{G}^*$, i.e. Player s loses $\mathcal{G}^*$ from $v^\uparrow$.

Consider a positional strategy σ of Player s in $\mathcal{G}$ and a play π^* in $\mathcal{G}^*$ from $v_0^\uparrow$ which is conform with σ^*. If π^* is infinite, it is of the form $\pi^* = \pi_{v_0 v_1}\pi_{v_1 v_2}\dots$ where $\pi = v_0 v_1 \dots$ is a play in $\mathcal{G}$ conform with σ and both plays have the same winner. If π^* is finite, it can be decomposed as $\pi^* = \pi_{v_0 v_1}\pi_{v_1 v_2}\dots\pi_{v_{n-1}v_n}\pi'$ where π' has no prefix of the form π_{vw}. In this case $v_0 \dots v_n$ is a finite play in $\mathcal{G}$ conform with σ and both plays are lost by the owner of v_n. Hence, if σ is winning from v_0, then σ^* is winning from $v_0^\uparrow$. $\square$

The simulation is illustrated in Figure 4 using two colors. Using the notion from [21] we can state that $\mathcal{G}^*$ has *strong tree-width* $\leq 4k$, which implies a tree-width bound of $8k-1$. This proves that parity games of clique-width $\leq k$ are log-space reducible to parity games of tree-width $\leq 8k-1$, under the assumption that a k-expression is provided, and hence Theorem 13 follows.

The algorithm by Obdržálek for parity games of bounded clique-width in [4] uses the fact that every winning strategy can be transformed into an equivalent t-*strategy*, which is a simple corollary of Lemma 14: By the positional determinacy theorem we can assume that Player 0 uses a positional strategy in the tree game $\mathcal{G}^*$, which indeed defines a t-strategy in $\mathcal{G}$.

References

1. C. Stirling, Local model checking games, in: I. Lee, S. A. Smolka (Eds.), CONCUR, Vol. 962 of LNCS, Springer, 1995, pp. 1–11.
2. M. Jurdziński, Deciding the winner in parity games is in UP $\cap$ co-UP, Information Processing Letters 68 (3) (1998) 119–124.

3. J. Obdržálek, Fast mu-calculus model checking when tree-width is bounded, in: W. A. H. Jr., F. Somenzi (Eds.), CAV, Vol. 2725 of LNCS, Springer, 2003, pp. 80–92.
4. J. Obdržálek, Clique-width and parity games, in: J. Duparc, T. A. Henzinger (Eds.), CSL, Vol. 4646 of LNCS, Springer, 2007, pp. 54–68.
5. J. Fearnley, S. Schewe, Time and parallelizability results for parity games with bounded treewidth, in: A. Czumaj, K. Mehlhorn, A. M. Pitts, R. Wattenhofer (Eds.), Automata, Languages, and Programming, Vol. 7392 of LNCS, Springer, 2012, pp. 189–200.
6. I. H. Sudborough, On the tape complexity of deterministic context-free languages, Journal of the Association for Computing Machinery 25 (3) (1978) 405–414.
7. S. Göller, M. Lohrey, Fixpoint logics over hierarchical structures, Theory Comput. Syst. 48 (1) (2011) 93–131.
8. H. Vollmer, Introduction to Circuit Complexity, Texts in Theoretical Computer Science, Springer, 1999.
9. E. A. Emerson, C. S. Jutla, Tree automata, mu-calculus and determinacy (extended abstract), in: 32nd Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1991, pp. 368–377.
10. I. Walukiewicz, Monadic second order logic on tree-like structures, in: C. Puech, R. Reischuk (Eds.), STACS 96, Vol. 1046 of LNCS, Springer, 1996, pp. 401–413.
11. M. Elberfeld, A. Jakoby, T. Tantau, Logspace versions of the theorems of Bodlaender and Courcelle, in: 51st Annual IEEE Symposium on Foundations of Computer Science, 2010, pp. 143–152.
12. O. Friedmann, M. Lange, Solving parity games in practice, in: Z. Liu, A. P. Ravn (Eds.), Automated Technology for Verification and Analysis, Vol. 5799 of LNCS, Springer, 2009, pp. 182–196.
13. B. Courcelle, Graphs as relational structures: An algebraic and logical approach, in: H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.), Graph-Grammars and Their Application to Computer Science, Vol. 532 of LNCS, Springer, 1990, pp. 238–252.
14. B. Courcelle, J. A. Makowsky, U. Rotics, Linear time solvable optimization problems on graphs of bounded clique-width, Theory Comput. Syst. 33 (2) (2000) 125–150.
15. S. Arnborg, D. G. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebraic Discrete Methods 8 (2) (1987) 277–284.
16. R. G. Downey, M. R. Fellows, Parameterized Complexity, Monographs in Computer Science, Springer Verlag, 1999.
17. B. Courcelle, S. Olariu, Upper bounds to the clique width of graphs, Discrete Applied Mathematics 101 (1-3) (2000) 77–114.
18. M. R. Fellows, F. A. Rosamond, U. Rotics, S. Szeider, Clique-width is NP-complete, SIAM J. Discrete Math. 23 (2) (2009) 909–939.
19. M. Lohrey, On the parallel complexity of tree automata, in: A. Middeldorp (Ed.), RTA, Vol. 2051 of LNCS, Springer, 2001, pp. 201–215.
20. E. Grädel, W. Thomas, T. Wilke (Eds.), Automata, Logics, and Infinite Games: A Guide to Current Research, Vol. 2500 of LNCS, Springer, 2002.
21. D. Seese, Tree-partite graphs and the complexity of algorithms, in: L. Budach (Ed.), FCT, Vol. 199 of LNCS, Springer, 1985, pp. 412–421.