

Clique-Width and Parity Games

Jan Obdržálek*

Faculty of Informatics, Masaryk University, Brno, Czech Republic
obdrzalek@fi.muni.cz

Abstract. The question of the exact complexity of solving parity games is one of the major open problems in system verification, as it is equivalent to the problem of model-checking the modal μ -calculus. The known upper bound is $\text{NP} \cap \text{co-NP}$, but no polynomial algorithm is known. It was shown that on tree-like graphs (of bounded tree-width and DAG-width) a polynomial-time algorithm does exist. Here we present a polynomial-time algorithm for parity games on graphs of bounded clique-width (class of graphs containing e.g. complete bipartite graphs and cliques), thus completing the picture. This also extends the tree-width result, as graphs of bounded tree-width are a subclass of graphs of bounded clique-width. The algorithm works in a different way to the tree-width case and relies heavily on an interesting structural property of parity games.

1 Introduction

Parity games are infinite two-player games of perfect information played on directed graphs where the vertices are labelled by priorities. The players take turns in pushing token along the edges of the graph. The winner is determined by the parity of the highest priority occurring infinitely often in this infinite play. The significance of parity games comes from the area of model-checking and system verification. The modal μ -calculus, a fixed-point logic of programs, was introduced by Kozen in [13] and subsumes most of the used modal and temporal logics. The problem of determining, given a system \mathcal{A} and a formula φ of the modal μ -calculus, whether or not \mathcal{A} satisfies φ can be translated to the problem of solving a parity game – see e.g. [9, 16] (the converse is also true).

The exact complexity of solving parity games is a long-standing open problem which received a large amount of attention. The best known upper bound is $\text{NP} \cap \text{co-NP}$ (more precisely $\text{UP} \cap \text{co-UP}$ [10]), but no polynomial-time algorithm is known. Up till recently the best known deterministic algorithm was the small progress measures algorithm [11] with time complexity $\mathcal{O}(dm \cdot (2n/d)^{d/2})$, where n is the number of vertices, m the number of edges and d the number of priorities. The first truly sub-exponential algorithm was the randomised algorithm by Björklund et al. [3], with the complexity bounded by $2^{\mathcal{O}(\sqrt{n \log n})}$. Recently Jurdziński et al. [12] came with a very simple deterministic sub-exponential algorithm of complexity roughly matching the upper bound of the randomised

* Supported by the Institute of Theoretical Computer Science, project 1M0545.

algorithm. Finally there is the strategy improvement algorithm of Vöge and Jurdziński [19] (and some others - e.g. [16]) which behaves extremely well in practice, but for which we do not have better than exponential upper bound.

Since so far we have not been able to find a polynomial-time algorithm for solving parity games on general graphs, the question is whether we can do better for some restricted classes of graphs. In [15] the author showed that there is a polynomial-time algorithm for solving parity games on graphs of bounded tree-width. (Tree-width, introduced by Robertson and Seymour in [18], is a well known and phenomenally successful connectivity measure for undirected graphs. Check [16] for a more elegant proof.) The result of [15] does not follow from the general result of Courcelle [4] which states that for a fixed formula φ of MSO and a graph of bounded tree-width the model-checking problem can be solved in linear time in the size of the graph. The important difference is that [4] considers the formula to be fixed, whereas in parity games the size of the formula describing the winning condition depends on the number of priorities, which can be as high as the number of vertices. (The constant factor of the algorithm presented in [4] depends on both n and k , and it is not even elementary [8]).

The way the concept of tree-width is applied to directed graphs is that we forget the orientation of the edges. That unfortunately means that tree-width can be high even for a very simple directed graphs (e.g. directed cliques). Therefore a new connectivity measure called DAG-width has been developed [17, 1] to better describe the connectivity of directed graphs. In the latter paper it was shown that on graphs of bounded DAG-width parity games can be solved in polynomial time. Another class of directed graphs for which we have a polynomial-time algorithm are graphs of bounded entanglement [2].

Both tree-width and DAG-width measure how close a graph is to a relatively simple/sparse graph (e.g. tree, DAG). Clique-width, defined in [6], stands on the opposite end of the spectrum – it measures how close is a graph to a complete bipartite graph (e.g. complete bipartite graphs and cliques have clique-width 2). In [6] it was also shown that bounded tree-width implies bounded clique-width, and so clique-width can be seen both as a generalisation and as a “dual notion” to tree-width. In this paper we present a polynomial-time algorithm for solving parity games on graphs of bounded clique-width. As argued above this both extends and complements our understanding of the complexity of solving parity games. As is the case for tree-width (see above), the result is not a consequence of a general result saying that MSO logic is decidable in linear time on graphs of bounded clique-width [5]. The reasons are the same as for tree-width.

The work is organised as follows: In Section 2 we present parity games, and prove a general property of parity games which is in the core our algorithm (but can be likely of use elsewhere). In Section 3 we present the definition of clique-width and some extra notation. Section 4 then contains the main result – the algorithm for solving parity games on graphs of bounded clique-width.

2 Parity Games

A *parity game* $\mathcal{G} = (V, E, \lambda)$ consists of a finite directed graph $G = (V, E)$, where V is a disjoint union of V_0 and V_1 (we assume that this partition is implicit), and a parity function $\lambda : V \rightarrow \mathbb{N}$ (we assume $0 \notin \mathbb{N}$). As it is usually clear from the context, we sometimes talk about a parity game G – i.e. we identify the game with its game graph. For technical reasons we also assume that the edge relation $E : V \times V$ is total: that is, for all $u \in V$ there is $v \in V$ such that $(u, v) \in E$. The game \mathcal{G} is played by two players P_0 and P_1 (also called EVEN and ODD), who move a single token along edges of the graph G . The game starts in an initial vertex and players play indefinitely as follows: if the token is on a vertex $v \in V_0$ ($v \in V_1$), then P_0 (P_1) moves it along some edge $(v, w) \in E$ to w . As a result, a play of \mathcal{G} is an infinite path $\pi = \pi_1\pi_2\dots$, where $\forall i > 0. (\pi_i, \pi_{i+1}) \in E$.

Let $\text{Inf}(\pi) = \{v \in V \mid v \text{ appears infinitely often in } \pi\}$. Player P_0 *wins the play* π if $\max\{\lambda(v) \mid v \in \text{Inf}(\pi)\}$ is even, and otherwise player P_1 wins. A (*total*) *strategy* σ (τ) for P_0 (P_1) is a function $\sigma : V^*V_0 \rightarrow V$ ($\tau : V^*V_1 \rightarrow V$) which assigns to each play $\pi.v \in V^*V_0$ ($\in V^*V_1$) a vertex w such that $(v, w) \in E$. A player *uses a strategy* σ in the play $\pi = \pi_1\pi_1\dots\pi_k\dots$, if $\pi_{k+1} = \sigma(\pi_1\dots\pi_k)$ for each vertex $\pi_k \in V_i$. A strategy σ is *winning* for a player and a vertex $v \in V$ if she wins every play that starts from v using σ .

If we fix an initial vertex v , then we say player P_i *wins the game* $\mathcal{G}(v)$ if he has a strategy σ such that using σ he wins every play starting in v . Finally we say that player *wins the game* \mathcal{G} if he has a strategy σ such that using σ he wins the game $\mathcal{G}(v)$ for each $v \in V$. By *solving* the game \mathcal{G} we mean finding the winner of $\mathcal{G}(v)$ for each vertex $v \in V$.

A *memoryless strategy* σ (τ) for P_i ($i \in \{0, 1\}$) is a function $\sigma : V_0 \rightarrow V$ ($\tau : V_1 \rightarrow V$) which assigns each $v \in V_i$ a vertex w such that $(v, w) \in E$. I.e. memoryless strategies do not consider the history of the play so far, but only the vertex the play is currently in. We use Σ_i to denote the set of memoryless strategies of player P_i , and $\overline{\Sigma}_i$ the sets of all strategies. For a strategy $\sigma \in \Sigma_0$ and $(v, w) \in V_0 \times V$ we define a strategy $\sigma[v \rightarrow w]$ which agrees with σ on all vertices except for v by the prescription

$$\sigma[v \rightarrow w](x) = \begin{cases} w & \text{if } x = v \\ \sigma(x) & \text{otherwise} \end{cases}$$

Parity games are memorylessly determined. By that we mean the following theorem.

Theorem 1 ([7, 14]). *For each parity game $\mathcal{G} = (V, E, \lambda)$ we can partition the set V into two sets W_0 and W_1 such that the player P_i has a memoryless winning strategy for all vertices in W_i .*

Example 1. Fig. 1 shows a parity game of six vertices. Circles denote the vertices of player P_0 and boxes the vertices of player P_1 . Priorities are written inside vertices. In this game player P_0 can win from the shaded vertices by forcing a

play to the vertex with priority four. Player P_1 has no choice in that vertex and must play to the vertex with priority three. The play will stay in the cycle with the highest priority four and therefore P_0 wins.

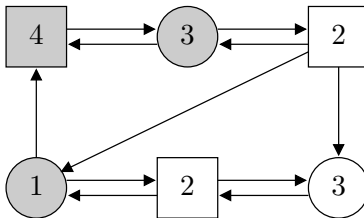


Fig. 1. A parity game

Definition 1 (G_σ). For game $\mathcal{G} = (V, E, \lambda)$ and a strategy $\sigma \in \Sigma_0$ we define G_σ to be the subgraph of G induced by σ - i.e. $G_\sigma = (V, E_\sigma)$, where $E_\sigma = E \setminus \{(v, w) \in E \mid v \in V_0 \text{ and } \sigma(v) \neq w\}$.

An immediate observation is that $\sigma \in \Sigma_0$ is winning for P_0 from v iff no cycle such that the highest priority on this cycle is odd is reachable from v in G_σ .

In addition to the standard ordering of priorities (by the relation “ $<$ ”), it is often useful to have priorities ordered from the point of their “attractiveness” for one of the players. I.e. for player P_0 a high even priority is more attractive than a low even one, which is still more attractive than any odd priority. We define the order \sqsubseteq in the following way:

Definition 2 (\sqsubseteq). For two priorities $p, q \in \mathbb{N}$ we write $p \sqsubset q$ if p is odd and v is even, or $p > q$ and p, q are odd, or $p < q$ and p, q are even. We write $p \sqsubseteq q$ if $p \sqsubset q$ or $p = q$.

We need a way of describing partial results for plays. More specifically we need to know the result for all paths between two specified vertices within a subgraph G' of G when a strategy σ of P_0 is fixed. Let $\Pi_\sigma(v, w, G')$ be the set of all paths in G'_σ from v to w . If $\Pi_\sigma(v, w, G') \neq \emptyset$ we define

$$\text{result}_\sigma(v, w, G') = \min_{\sqsubseteq} \{ \max \{ \lambda(\pi_i) \mid 0 \leq i \leq j \} \mid v = \pi_0, \pi_1, \dots, \pi_j = w \in \Pi_\sigma(v, w, G') \}$$

If $G' = G$ we write just $\text{result}_\sigma(v, w)$. We also write $x \rightarrow_\sigma y$ if $(x, y) \in E(G_\sigma)$, and use \rightarrow_σ^* to denote the reflexive and transitive closure of \rightarrow_σ .

2.1 Joint Choice Property

The following theorem is in the core of the proof of the main result of this paper. It basically says that if we have a set $U \subseteq V_0$ of vertices of P_0 such that they have the same sets of successors and a memoryless strategy σ winning for at least one of these vertices, then there is a memoryless strategy σ' which assigns to each $u \in U$ the same successor and is winning for all the vertices of U and all the vertices for which σ is a winning strategy for P_0 .

Theorem 2. *Let $\mathcal{G} = (V, E, \lambda)$ be a parity game, $U \subseteq V_0$ a set of vertices of P_0 and $\sigma \in \Sigma_0$ a strategy of P_0 . Also denote $\sigma(U) = \bigcup_{u \in U} \sigma(u)$ and require that $\forall u \in U \forall v \in \sigma(U). (u, v) \in E$. If σ is a winning strategy for some vertex $w \in U$ and P_0 , then there exists a vertex \bar{u} in $\sigma(U)$ such that the strategy σ' defined as*

$$\sigma'(v) = \begin{cases} \bar{u} & \text{if } v \in U \\ \sigma(v) & \text{otherwise} \end{cases}$$

is a winning strategy for P_0 from all vertices of U . Moreover for all $v \in V(G)$ if σ is winning for v and P_0 , so is σ' .

Proof. First note that we can assume that σ is winning for all vertices of U . If it is not, we can simply switch the strategy into any winning successor. Let $\sigma \in \Sigma_0$ be fixed and let us take a set $U \subseteq V_0$ satisfying the requirements above. We will proceed by induction on the size of U . If U is a set with a single vertex u , then we just put $\sigma' = \sigma$ as σ must be a winning strategy for P_0 and u .

Therefore assume $|U| = k + 1$ for some $k \in \mathbb{N}$. Let $U = X \cup \{x'\}$, where $X \subset U$ has k elements. By induction hypothesis there exists $y \in \sigma(X)$ such that the strategy $\bar{\sigma}$ defined as $\bar{\sigma}(x) = y$ for $x \in X$ and identical with σ on all other vertices is winning for P_0 from all vertices of X .

If $\sigma(x') = \bar{\sigma}(x') = y$ we put $\sigma' = \bar{\sigma}$ and we are done, since $\bar{\sigma}$ is also winning from x' and satisfies the requirements. Otherwise $y' = \bar{\sigma}(x') \neq y$ and there are three cases to be considered (see Fig. 2):

1. If $y \not\rightarrow_{\bar{\sigma}}^* x'$ we can safely set $\sigma' = \bar{\sigma}[x' \rightarrow y]$, as this change of strategy cannot introduce a new cycle to $G_{\bar{\sigma}}$. Obviously σ' is winning for x' as $\bar{\sigma}$ is winning for y .
2. If $y \rightarrow_{\bar{\sigma}}^* x'$ but $y' \not\rightarrow_{\bar{\sigma}}^* x$ for all $x \in X$ we can similarly put $\sigma' = \bar{\sigma}_{x \in X}[x \rightarrow y']$.
3. The last case is there are both a path from y to x' and a set of paths from y' to X in the graph $G_{\bar{\sigma}}$. As in the previous case $\bar{\sigma}$ is winning for both y and y' , and therefore no odd cycle is reachable from y and y' in $G_{\bar{\sigma}}$. Let us put $p_1 = \text{result}_{\bar{\sigma}}(y, x')$ and $p_2 = \min_{\sqsubseteq} \bigcup_{x \in X} \text{result}_{\bar{\sigma}}(y', x)$. Note that $p = \max(p_1, p_2)$ must be even, otherwise there would be an odd cycle through x' in $G_{\bar{\sigma}}$, which is not possible since $\bar{\sigma}$ is winning for all $x \in X$. Now if $p = p_1$ put $\sigma' = \bar{\sigma}[x' \rightarrow y]$, otherwise $\sigma' = \bar{\sigma}_{x \in X}[x \rightarrow y']$.

Let us assume the first case ($p = p_1$). Then all newly created cycles must use the edge (x, y') , and for the highest priority p on such a cycle it must be the case that $p_1 \sqsubseteq p$. By definition of p_1 all the newly created cycles are winning for P_0 . The case $p = p_2$ is similar.

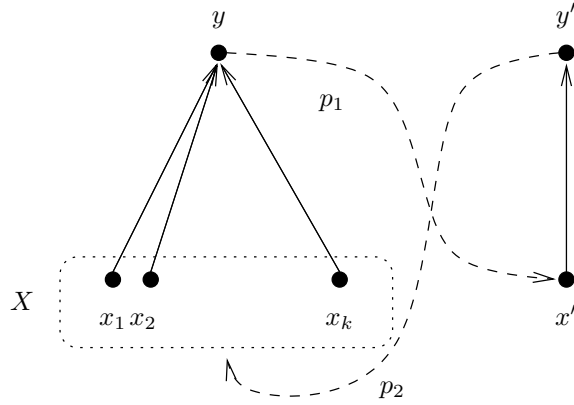


Fig. 2. Illustration of the proof of Theorem 2

3 Clique Width

The notion of clique-width was first introduced by Courcelle and Olariu in [6]. There are actually two definitions of clique-width in that paper, one for directed graphs and one for undirected graphs. We will present the former definition, as it is more suitable for parity games, graphs of which are directed.

Let k be a positive integer. We call (G, γ) a k -graph if G is a graph and $\gamma : V(G) \rightarrow \{1, 2, \dots, k\}$ is a mapping. (As γ is usually fixed, we often write just G for the k -graph (G, γ) .) We call $\gamma(v)$ for $v \in V(G)$ the *label* of a vertex v (also the *colour* of v). We also define the following operators:

1. For $i \in \{1, 2, \dots, k\}$ let $[i]$ denote an isolated vertex labelled by i .
2. For $i, j \in \{1, 2, \dots, k\}$ with $i \neq j$ we define a unary operator $\alpha_{i,j}$ which adds edges between all pairs of vertices with label i and j . Formally

$$\alpha_{i,j}(G, \gamma) = (G', \gamma)$$

where $V(G') = V(G)$ and $E(G') = E(G) \cup \{(v, w) \mid v, w \in V, \gamma(v) = i \text{ and } \gamma(w) = j\}$.

3. For $i, j \in \{1, 2, \dots, k\}$ let $\rho_{i \rightarrow j}$ be the unary operator which relabels every vertex labelled by i to j . Formally

$$\rho_{i \rightarrow j}(G, \gamma) = (G, \gamma')$$

where

$$\gamma'(v) = \begin{cases} j & \text{if } \gamma(v) = i \\ \gamma(v) & \text{otherwise} \end{cases}$$

4. Finally \oplus is a binary operation which makes a disjoint union of two k -graphs. Formally

$$(G_1, \gamma_1) \oplus (G_2, \gamma_2) = (G, \gamma)$$

where $V(G) = V(G_1) \uplus V(G_2)$, $E(G) = E(G_1) \uplus E(G_2)$ and

$$\gamma(v) = \begin{cases} \gamma_1(v) & \text{if } v \in V(G_1) \\ \gamma_2(v) & \text{if } v \in V(G_2) \end{cases}$$

A *k-expression* is a well formed expression t written using the operators defined above. The k -graph produced by performing these operations in order therefore has as a vertex set the set of all occurrences of the constant symbols in t , and this k -graph (or any k -graph isomorphic to it) is called the *value* $val(t)$ of t . If a k -expression t has value (G, γ) we say that t is a *k-expression corresponding to G*. The *clique-width* of G , written as $cwd(G)$, is the minimum k such that there is a k -expression corresponding to G .

Example 2. $\alpha_{1,2}(\rho_{1 \rightarrow 2}(\alpha_{1,2}(\rho_{1 \rightarrow 2}(\alpha_{1,2}([1] \oplus [2]))) \oplus [1])) \oplus [1]$ is a 2-expression corresponding to a directed clique of size 4. See Fig. 3.

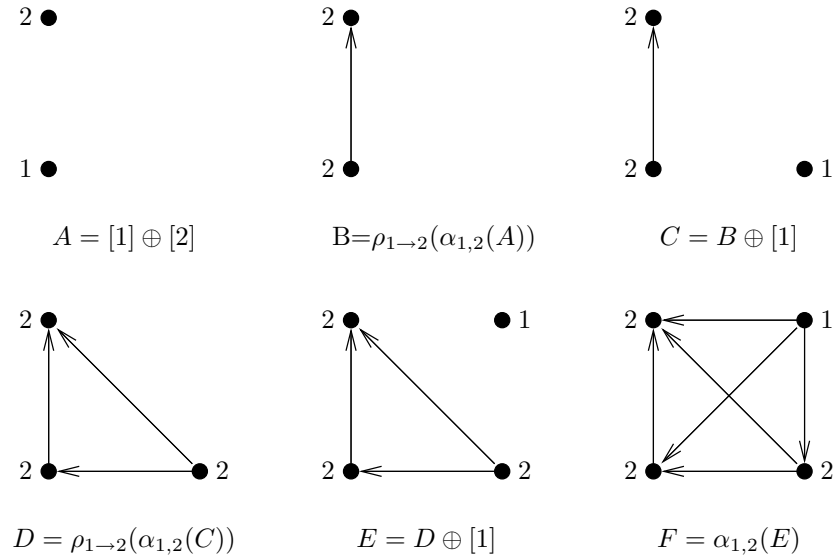


Fig. 3. Construction of the directed clique of size 4

Note aside: For undirected graphs the clique-width is defined in just the same way, except for the operator $\alpha_{i,j}$. In undirected case it is replaced by an operator $\eta_{i,j}$ which creates undirected edges between all vertices with colours i and j .

It is quite natural to view a k -expression t_G corresponding to G as a tree T with nodes labelled by subterms of t_G (t_G is the root), together with a bijection between the leaves of the tree and vertices of G . In this setting the *type* of each node $t \in V(T)$ is the top-level operator of t , so we have four different node types.

In the rest of this paper we will always have G and a k -term t_G (and therefore also its tree T) corresponding to G fixed. We will also use the following notation: For a node $t \in V(T)$ let $G[t]$ be the subgraph of G given by t , and $V[t]$ ($E[t]$) the set of its vertices (edges). Slightly abusing the notation we use $\gamma(v, t)$ to denote the colour of v in the node t , $\gamma(i, t)$ to denote the set of vertices of G which have the colour i at the node t , and $\gamma(t)$ to denote the set of all colours whose associated vertex sets for t contain at least one vertex.

3.1 k -Expressions and t -Strategies

For the purposes of our algorithm we have to consider a special kind of memoryless strategies, called t -strategies. Such strategies assign to all free (see below) even vertices of the same colour the same successor. Let $\mathcal{G} = (V, E, \lambda)$ be a parity game, where $G = (V, E)$ is of clique-width k and t is the corresponding k -expression (and T its associated tree). A vertex $v \in \gamma(i, t)$ is *free* for $\sigma \in \Sigma_0$ if $(v, \sigma(v)) \notin E[t]$. A strategy $\sigma \in \Sigma_0$ is called a t -strategy if for each $t' \in V(T)$ of type $\alpha_{i,j}(t')$ we have that for all free $v, w \in \gamma(i, t') \cap V_0$ it is true that $\sigma(v) = \sigma(w)$. We use Σ_0^t to denote the set of all t -strategies.

Theorem 3. *Let $\mathcal{G} = (V, E, \lambda)$ be a parity game, and t the corresponding k -expression to $G = (V, E)$. Let $\sigma \in \Sigma_0$ be a strategy winning for P_0 from $W \subseteq V$. Then there is a t -strategy $\sigma' \in \Sigma_0^t$ such that σ' is winning from W .*

Proof (sketch). By repeated application of Theorem 2 at nodes of type $\alpha_{i,j}(t')$ following the structure of t from the leaves upwards. Let $X \subseteq \gamma(i, t')$ be the set of all free vertices for σ . Then from now on they can only be connected to the same common successors. Note that edges from X to $\sigma(X)$ must be added either in this node or somewhere higher up the tree T . If σ is winning for some $x \in X$, then we can apply the Theorem 2 to find the common winning successor \bar{u} , and change the strategy σ . If this \bar{u} is in $\gamma(j, t')$ we are done for this node. Otherwise we deal with X higher up in T . If σ is not winning for any vertex of X , we select a common successor at random.

4 Algorithm

In this section we present the main result – that parity games can be solved in polynomial time on graphs of bounded clique-width. The algorithm described here is in spirit similar to the one for graphs of bounded tree-width [15] and bounded DAG-width [1, 16]. However there are many conceptual differences, as there is no small set of vertices forming an *interface* between an already processed subgraph and the rest of the graph. To the contrary, in one step we can connect an unbounded number of edges of G . The problem is we cannot keep the results for all the individual vertices of one colour. This is the main obstacle we have to deal with.

For the rest of this section let us fix a parity game $\mathcal{G} = (V, E, \lambda)$, where $G = (V, E)$ is of clique-width k and t_G is the corresponding k -expression (and

T its associated tree). In addition we will assume that \mathcal{G} is bipartite - i.e. that $E \subseteq (V_0 \times V_1) \cup (V_1 \times V_0)$. This requirement implies that for each $t \in V(T)$ and each colour i we have that either $\gamma(i, t) \subseteq V_0$ or $\gamma(i, t) \subseteq V_1$. (If vertices of both players have the same colour, then there cannot be any edge added to or from this colour.) We therefore talk about *even colours* – containing the vertices from V_0 , and similarly of *odd colours*. This will allow us to simplify our construction. At the end of this section we discuss how we can modify the construction so it does not require this assumption.

In the previous section we have seen the definition of $result_\sigma$ for the set of all paths between two vertices in a subgraph G' of G . We need to extend this definition to colours and subgraphs $G[t]$ for $t \in V(T)$.

For a vertex $v \in V[t]$ and strategy $\sigma \in \overline{\Sigma_0}$ we want to find the best result the player P_1 can achieve against this strategy for a play starting from v within $G[t]$. If there is a winning cycle of P_1 (i.e. the highest priority on this cycle is odd) reachable from v in the game $G[t]$ when P_0 uses σ , then we know P_1 can win against the strategy σ if starting in v in the game G . Falling short of this objective the player P_1 can force a play to some vertex $w \in \gamma(i, t)$, with the hope of extending this play to a winning play as more edges and vertices are added. The ‘value’ of such play is the highest priority of a vertex on this path. However note that there can be more paths starting in v which lead to a vertex of a given colour i . In that case it is in the player P_1 ’s interest to choose the one with the lowest score w.r.t. the ‘ \sqsubseteq ’ ordering. For each colour in $\gamma(t)$ we remember the best such result. But not all vertices w can be used to extend a play. As σ is fixed, only vertices from V_1 and those vertices of V_0 at the end of a maximal finite path (i.e. free vertices) qualify.

To formalise the description above, we need to extend the ‘ \sqsubseteq ’ ordering to pairs (i, p) . For two such pairs $(i, p), (j, q)$ we put $(i, p) \sqsubseteq (j, q)$ iff $i = j$ and $p \sqsubseteq q$. Specifically if $i \neq j$ then the two pairs are incomparable. We extend the ordering \sqsubseteq by adding the minimal element \perp . For a set $X \subseteq (\{1, \dots, k\} \times \mathbb{N}) \cup \{\perp\}$ we denote $\min_{\sqsubseteq} X$ to be the set of \sqsubseteq -minimal elements of X . Note that $\min_{\sqsubseteq} X = \{\perp\}$ iff $\perp \in X$. Moreover for $Z = \min_{\sqsubseteq} X$ if $Z \neq \{\perp\}$ then Z contains at most k pairs (i, p) , one for each colour i .

We consider the maximal plays first. Starting from a vertex $v \in V[t]$ and given a strategy $\sigma \in \overline{\Sigma_0}$ let Π be the set of maximal paths in $G[t]$ using σ . For a finite path $\pi = \pi_0, \dots, \pi_j$ we define the operator $r(\pi) = (\gamma(\pi_j), \max\{\lambda(\pi_i) \mid 0 \leq i \leq j\})$. Then we put

$$result_\sigma(v, G[t]) = \begin{cases} \perp & \text{if } \exists \pi \in \Pi \text{ s.t. } \pi \text{ is infinite and winning for } P_1 \\ \min_{\sqsubseteq} \bigcup_{\pi \in \Pi} r(\pi) & \text{otherwise} \end{cases}$$

To the set defined above we add also the results of all finite plays in $G[t]$ starting in v and ending in a vertex in V_1 (the set X). Finally we take only the minimal elements.

$$\begin{aligned}
\text{result}_\sigma(v, t) &= \min_{\sqsubseteq} (X \cup Y) \\
X &= \{(\gamma(w), \text{result}_\sigma(v, w, G[t])) \mid w \in V_1 \cap V[t]\} \\
Y &= \text{result}_\sigma(v, G[t])
\end{aligned}$$

Note: The set $\text{result}_\sigma(v, t)$ will never be empty for $v \in V[t]$. Even when all the reachable cycles are won by P_0 , there must be vertices in $V_1 \cap V[t]$ reachable from v since the game is bipartite.

Finally we want to say what happens when the play is restricted to $G[t]$ and starts in any vertex of colour i . The definition of $\text{result}_\sigma(i, t)$ depends on whether i is odd or even colour. The reason for this is the following: if i is an odd colour, then all edges to vertices of colour i are from even vertices. Using a memoryless strategy the player P_0 chooses *one* vertex of colour i as a successor. On the other hand if i is an even colour, then the player P_1 , playing against σ , can choose *any* vertex of colour i in the next step. This is formalised as follows:

$$\text{result}_\sigma(i, t) = \begin{cases} \{\text{result}_\sigma(v, t) \mid v \in \gamma(i, t)\} & \text{if } i \text{ is odd colour in } t \\ \min_{\sqsubseteq} \bigcup_{v \in \gamma(i, t)} \text{result}_\sigma(v, t) & \text{if } i \text{ is even colour in } t \end{cases}$$

For $t \in V(T)$ and $i \in \gamma(t)$ we define the set $\text{Result}(i, t)$ as the set of all possible results when starting in a (vertex of) colour i and restricted to $G[t]$.

$$\text{Result}(i, t) = \{\text{result}_\sigma(i, t) \mid \sigma \in \overline{\Sigma_0}\}$$

Note that in the definition above we do not require σ to be memoryless.

In our algorithm we use dynamic programming on T to compute sets $R(i, t)$ for each node t of T from the bottom up. We distinguish four cases (as we have four types of nodes). For each node we have to prove (by induction from the leaves) that the following two properties hold:

- P1. $\forall i \in \gamma(t). R(i, t) \subseteq \text{Result}(i, t)$
- P2. $\{\text{result}_\sigma(i, t) \mid \sigma \in \Sigma_0^t\} \subseteq R(i, t)$

The property P1 states that each member of $R(i, t)$ is a result for some strategy $\sigma \in \overline{\Sigma_0}$. The property P2 then means that the results for all t -strategies are included.

The following lemma states that if the sets $R(i, t)$ satisfy properties P1 and P2, then we can effectively solve the game $\mathcal{G}(v)$. Here we assume that v is the only vertex of colour i , it is created by the $[i]$ operator, and keeps its colour all the time till the end of the construction. It is not difficult to modify t (by adding an extra colour) so it satisfies this requirement.

Lemma 1. *Let $\mathcal{G} = (V, E, \lambda)$ be a parity game where $G = (V, E)$ is of clique-width k and t is the associated k -expression corresponding to G . Let $R(i, t)$ be a set satisfying P1 and P2, and $\gamma(i, t) = \{v\}$. Then P_0 wins the game starting in v iff $R(i, t) \neq \{\perp\}$.*

Proof. Let $\sigma \in \Sigma_0$ be a winning strategy of P_0 from v . By Theorem 3 there must be $\sigma' \in \Sigma_0^t$ such that σ' is also winning from v . By P2 $result_{\sigma'}(i, t) \in R(i, t)$ and by definition of $result_{\sigma'}(i, t)$ we must have $result_{\sigma'}(i, t) \neq \{\perp\}$.

On the other hand if $R(i, t) \neq \{\perp\}$, then by P1 there must be a strategy σ s.t. $result_{\sigma}(i, t) = result_{\sigma}(v, t) \neq \{\perp\}$. By definition of $result_{\sigma}(v, t)$ there is no odd cycle reachable from v if P_0 is using σ and therefore P_0 wins the game G for v .

4.1 Algorithms for Operators

Here we present the algorithms for all four types of nodes. Once the algorithm is given the proof is just a simple, if tedious, work and follows from the definition of $result_{\sigma}$. To keep the presentation clear and tidy we only sketch some of the proofs.

t is of type $[i]$ Let v be the corresponding vertex of G . Then we put $R(i, t) = \{(i, \lambda(v))\}$. In this case obviously $R(i, t) = Result(i, t)$ and therefore both P1 and P2 hold.

t is of type $\rho_{i \rightarrow j}(t')$ We have to deal with two separate issues: choose the “better” result (for P_1) for paths ending in i and j and “join” $R(i, t')$ and $R(j, t')$. To address the first issue we define the operation *mod* which takes $\alpha \in R(l, t)$ and works as follows: If α contains both (i, p_i) and (j, p_j) , it replaces these two pairs by a single pair $(j, \min_{\sqsubseteq}(p_i, p_j))$ (and if α contains only a pair (i, p) , it is replaced by (j, p)).

$$mod(\alpha) = \begin{cases} \min_{\sqsubseteq}((\alpha \cup \{(j, p)\}) \setminus \{(i, p)\}) & \text{if } (i, p) \in \alpha \text{ for some } p \\ \alpha & \text{otherwise} \end{cases}$$

The sets $R(l, t)$ for $l \neq i, j$ are then defined in a straightforward way:

$$R(l, t) = \{mod(\alpha) \mid \alpha \in R(l, t')\}$$

That $R(l, t)$ satisfies P1 and P2 follows from the fact that $R(l, t')$ does and the definition of $result_{\sigma}(l, t)$.

Now it remains to show how to compute the set $R(j, t)$. Let $S_i = \{mod(\alpha) \mid \alpha \in R(i, t')\}$ and $S_j = \{mod(\beta) \mid \beta \in R(j, t')\}$ the sets modified as above. To produce the set $R(j, t)$ we have to merge S_i and S_j . The way we do this depends on whether i, j are even or odd colours (by definition they must be either both odd or both even).

$$R(j, t) = \begin{cases} \{\min_{\sqsubseteq}(\alpha \cup \beta) \mid \alpha \in S_i, \beta \in S_j\} & \text{if } i, j \text{ are even} \\ S_i \cup S_j & \text{if } i, j \text{ are odd} \end{cases}$$

Proposition 1. $R(j, t)$ satisfies P1 and P2.

Proof. For odd colours i, j P1 follows from the definition of $result_\sigma$. For P2 let $\sigma \in \Sigma_0^t$. By P2 for t' we have $result_\sigma(i, t') \in R(i, t')$ and $result_\sigma(j, t') \in R(j, t')$. The rest follows from the fact that $\gamma(j, t) = \gamma(i, t') \cup \gamma(j, t')$.

For even i, j take $\alpha \in S_i$ and $\beta \in S_j$. Then by induction hypothesis $\alpha = mod(result_{\sigma_1}(i, t'))$ for some σ_1 and $\beta = mod(result_{\sigma_2}(j, t'))$ for some σ_2 . Let σ be a strategy which behaves like σ_1 if we start in a vertex of $\gamma(i, t')$ and behaves like σ_2 if we start in a vertex of $\gamma(j, t')$. Then $result_\sigma(j, t) = \min_{\sqsubseteq}(\alpha \cup \beta)$. This proves P1.

For P2 and a strategy $\sigma \in \Sigma_0^t$ take $\alpha = result_\sigma(i, t')$ and $\beta = result_\sigma(j, t')$. By P2 for t' we have $\alpha \in R(i, t')$ and $\beta \in R(j, t')$. The rest follows from the definition of $result_\sigma$.

t is of type $\alpha_{i,j}(t')$ We are adding edges between colours i and j in this step. We start with computing the set $R(j, t)$ first. For each $\alpha \in R(j, t')$ we construct α' by taking $\alpha' = \alpha$ and looking for any new winning cycles for P_1 : If there is a pair $(i, p) \in \alpha$ we put \perp into α' if p is odd. We then chose the minimal pairs in α' – the set $\min_{\sqsubseteq}(\alpha')$ – and call the result $mod_1(\alpha)$. This works well for odd i . However for even i if we prolong the path through i we have in addition to remove the pair (i, p) (as there is not now any free vertex of colour i) – and we call such set $mod_2(\alpha)$. The last case is we do not add the edges between the vertices of colour i, j . Altogether:

$$R(j, t) = \begin{cases} \{mod_1(\alpha) \mid \alpha \in R(j, t')\} & \text{if } i \text{ is odd} \\ \{mod_2(\alpha) \mid \alpha \in R(j, t')\} \cup R(j, t') & \text{if } i \text{ is even} \end{cases}$$

Note: We do not add the set $R(j, t')$ if there is no $w \in V \setminus V[t]$ such that $(u, w) \in E$ for $u \in \gamma(i, t)$. The reason is that we cannot postpone the choice of strategy for free vertices in $\gamma(i, t)$ as there is no choice left.

Proposition 2. P1 and P2 hold for $R(j, t)$

Proof. For i odd (implies j is even) and P1 take $\alpha \in R(j, t')$. By P1 for t' $\alpha = result_\sigma(j, t')$ for some σ . But then $mod_1(\alpha) = result_\sigma(j, t)$ from the definition of $result_\sigma$.

So let i be even (this implies j is odd) and $\alpha \in R(j, t')$. By P1 for t' $\alpha = result_\sigma(v, t')$ for some σ and $v \in \gamma(i, t')$. For the first part of the union take σ' to be the strategy which behaves like σ until the play reaches a vertex in i , then goes to v and then continues again as σ . Then $result_{\sigma'}(i, t) = mod_2(\alpha)$. Finally for the second part consider σ' which behaves like σ and to free $v \in \gamma(i, t)$ it assigns a successor not in $V[t]$. Then $result_{\sigma'}(i, t) = \alpha$.

For P2 the proofs are very similar, the difference being that for each $\sigma \in \Sigma_0^t$ we take $\alpha = result_\sigma(i, t')$ to start with.

Once the set $R(j, t)$ is computed, the sets $R(l, t)$ for $l \in \gamma(t) \setminus \{j\}$ are computed in the following way. Take an element $\alpha \in R(l, t')$. If α contains a pair (i, p) for some p , then consider all possibilities of extending a path from l to i by an element of $R(j, t)$. The way of combining $R(l, t')$ and $R(j, t)$ is defined using the operator *weld*:

Definition 3 (*weld*). *Let $\alpha \in R(l, t')$, $\beta \in R(j, t)$ and $l \neq j$. Then*

$$\text{weld}_{i,j}(\alpha, \beta) = \begin{cases} \min_{\sqsubseteq} ((\alpha \cup \beta_p)) & \text{if } (i, p) \in \alpha \text{ for some } p \\ \alpha & \text{otherwise} \end{cases}$$

where $\beta_p = \{(k, \max\{p, q\}) \mid (k, q) \in \beta\}$.

Let $\text{weld}'_{i,j}$ be defined in exactly the same way, except for it removes the pair (i, p) from α . Finally we put

$$R(l, t) = \begin{cases} \{\text{weld}_{i,j}(\alpha, \beta) \mid \alpha \in R(l, t'), \beta \in R(j, t)\} & \text{if } i \text{ is odd} \\ \{\text{weld}'_{i,j}(\alpha, \beta) \mid \alpha \in R(l, t'), \beta \in R(j, t)\} \cup R(l, t') & \text{if } i \text{ is even} \end{cases}$$

Note: We do not add the set $R(l, t')$ if i is an even colour and there is no $w \in V \setminus V[t]$ such that $(u, w) \in E$ for $u \in \gamma(i, t)$. The reason is that we cannot postpone the choice of strategy for free vertices in $\gamma(i, t)$ as there is no choice left.

Proposition 3. *P1 and P2 hold for $R(l, t)$*

Proof (sketch). Similar to the proof of Proposition 2. For P1 let $\alpha \in R(l, t')$ and $\beta \in R(j, t)$. By P1 and the construction for $R(j, t)$ above we have $\alpha = \text{result}_{\sigma_1}(l, t')$ and $\beta = \text{result}_{\sigma_2}(j, t)$ for some σ_1, σ_2 . For the proof we take σ to be the strategy that behaves like σ_1 until it reaches i and like σ_2 afterwards.

For P2 and each $\sigma \in \Sigma_0^t$ we take $\alpha = \text{result}_{\sigma}(l, t')$ and $\beta = \text{result}_{\sigma}(j, t)$, which must be in $R(l, t')$ and $R(j, t)$ by induction hypothesis and construction. The rest follows from the definition of result_{σ} .

t is of type $t_1 \oplus t_2$ This node type is pretty straightforward. What we do depends on the colour – for a colour i we put

$$R(i, t) = \begin{cases} R(i, t_1) \cup R(i, t_2) & \text{if } i \text{ is an odd colour} \\ \{\min_{\sqsubseteq}(\alpha_1 \cup \alpha_2) \mid \alpha_1 \in R(i, t_1), \alpha_2 \in R(i, t_2)\} & \text{if } i \text{ is an even colour} \end{cases}$$

The proof that both P1 and P2 hold is follows immediately from the definition of $\text{result}_{\sigma}(i, t)$.

4.2 Complexity

Let us have a look at the size of the sets $R(i, t)$. First note that the set $\min_{\sqsubseteq} X$ for $X \subseteq (\{1, \dots, k\} \times \mathbb{N})$ can have at most k elements, each chosen from the set of available priorities, size of which can be bounded by $n = |V|$. Therefore the set $R(i, t)$ can contain at most $(n + 1)^k + 1$ different elements. As we have k different colours, the size of the data computed in each step is bounded by $k \cdot (n + 1)^k + 1$.

The number of steps is equal to the size (the number of operators) of the k -expression t corresponding to G (without loss of generality we can assume that $|t|$ is linear in $|V|$). Finally we see that the running time of the algorithms for different operators is at most the square of $|R(i, t)|$, as in some cases we have to take all pairs of members of $R(i, t)$. This, together with Lemma 1, proves the main result:

Theorem 4. *Let $\mathcal{G} = (V, E, \lambda)$ be a parity game where $G = (V, E)$ is of clique-width k and t is the associated k -expression corresponding to G . Then there is an algorithm which solves the parity game \mathcal{G} in time polynomial in n .*

4.3 Dropping the Restriction on Colours

In the algorithm above we restricted ourselves to bipartite parity games, where players alternate their moves. Here we briefly discuss how to drop this restriction.

The first step is to split each colour i into two colours i_0 and i_1 . The colour i_0 will be used only vertices from V_0 , and the colour i_1 similarly for vertices from V_1 . This at most doubles the number of colours. The second difference is that for each colour i_x we will keep two sets $R_0(i_x, t)$ and $R_1(i_x, t)$. One will be computed like it was a set for an even colour, and the other one as a set for odd colour. This again at most doubles the amount of information we need to keep. Altogether we store four times more information than in the basic algorithm.

The algorithms for the different node types are modified as follows. A vertex v created by the operator $[i]$ is added to the set it belongs to. For the \oplus operator we join separately the sets for even and odd versions of the same colour. For renaming $\rho_{i \rightarrow j}$ we again rename separately. Finally we have to deal with the $\alpha_{i,j}$ operator. The odd and even version of the i colour are treated separately. For i_0 we connect first to $R_1(j_0, t)$ and then to $R_1(j_1, t)$. Similarly for i_1 we connect first to $R_0(j_0, t)$ and then to $R_0(j_1, t)$.

Altogether the number of steps and the size of the sets we have to remember is within a constant factor of the original algorithm, so the running time remains polynomial.

References

1. D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. DAG-width and parity games. In *STACS'06*, volume 3884 of *LNCS*, pages 524–536. Springer-Verlag, 2006.

2. D. Berwanger and E. Grädel. Entanglement – a measure for the complexity of directed graphs with applications to logic and games. In *LPAR 2004*, volume 3452 of *LNCS*, pages 209–223. Springer-Verlag, 2004.
3. H. Björklund, S. Sandberg, and S. Vorobyov. A discrete subexponential algorithm for parity games. In *STACS 2003*, volume 2607 of *LNCS*, pages 663–674. Springer-Verlag, 2003.
4. B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B: Formal Models and Semantics*, pages 193–242. Elsevier, Amsterdam, 1990.
5. B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.
6. B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Appl. Math.*, 101(1-3):77–114, 2000.
7. E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. 5th IEEE Foundations of Computer Science*, pages 368–377, 1991.
8. M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *LICS 2002*, pages 215–224. IEEE Computer Society, 2002.
9. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer-Verlag, 2002.
10. M. Jurdziński. Deciding the winner in parity games is in $UP \cap co-UP$. *Information Processing Letters*, 68(3):119–124, 1998.
11. M. Jurdziński. Small progress measures for solving parity games. In *STACS 2000*, volume 1770 of *LNCS*, pages 290–301. Springer-Verlag, 2000.
12. M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pages 117–123. ACM-SIAM, 2006.
13. D. Kozen. Results on the propositional μ -calculus. *Theoretical Computer Science*, 27:333–354, 1983.
14. A. W. Mostowski. Games with forbidden positions. Technical Report 78, Instytut Matematyki, Uniwersytet Gdański, Poland, 1991.
15. J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *CAV 2003*, volume 2725 of *LNCS*, pages 80–92. Springer-Verlag, 2003.
16. J. Obdržálek. *Algorithmic Analysis of Parity Games*. PhD thesis, University of Edinburgh, 2006.
17. J. Obdržálek. DAG-width – connectivity measure for directed graphs. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pages 814–821. ACM-SIAM, 2006.
18. N. Robertson and P. D. Seymour. Graph Minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36:49–63, 1984.
19. J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *CAV 2000*, volume 1855 of *LNCS*, pages 202–215. Springer-Verlag, 2000.