Vol.2021-AL-182 No.8
2021/3/17

Parity-Game Reduction by Winning-Cycles

Ryota Tsukatani[1] Remy Belmonte[1] Hiro Ito[1]

Abstract: Parity games are games that are played on directed graphs, where each vertex is labeled with a natural number. The vertices consist of two disjoint subset V_0 and V_1. Two players P_0 and P_1 move the token along the edges of graph, starting from the given initial vertex. If the token is on a vertex in V_0, then P_0 moves the token along one of the out-going edges from the vertex; otherwise, P_1 does. If a player cannot move the token, he/she loses. Otherwise, the winner is determined by the maximum labeled number appearing infinitely often: if the number is even, P_0 wins; otherwise P_1 does. Deciding the winner in parity games belongs to NP and coNP, and it is open if it can be solved in polynomial-time. In this report, we introduce notions of forcing-cycles and winning-cycles, and give a polynomial-time algorithm for finding them. Moreover we show an algorithm to reduce a parity game by removing a winning-cycle if exists. By adapting this method, some parity games can be solved, or reduced to proper subgames in polynomial-time.

1. Introduction

Parity games are games that are played on directed graphs, where each vertex is labeled with a natural number, called priority. And the vertices consist of two disjoint subsets V_0 and V_1. Two players P_0 and P_1 move the token along the edges of graph, starting from the given initial vertex. If the token is on a vertex in V_0, then P_0 moves the token along one of the out-going edges from the vertex; otherwise, P_1 does. If a player cannot move the token, he/she loses. Otherwise, the winner is determined by the maximum priority appearing infinitely in the rout that the token follows: if the priority is even, P_0 wins; otherwise P_1 does.

Studying parity games has at least three important reasons. Firstly, the algorithmic problem of finding the winner in parity games is polynomial-time equivalent to the model checking problem of modal μ-calculus [6], which is a basic problem on algorithms in automated software verification. Secondly, parity games are polynomial-time reducible to other infinite games, e.g., mean-payoff games, discounted mean-payoff games, and simple stochastic games [4], [8]. Finally, its complexity status is intriguing: deciding the winner of parity games belongs to NP $\cap$ coNP, more precisely UP $\cap$ coUP [8], as well as in QP (quasi-polynomial time) [3], [14], yet no polynomial-time algorithm has not been given [12].

Algorithms for parity games started with McNaughton's algorithm, which was first given by McNaughton for muller games [10], and adapted to parity games by Zielonka [15]. While the running time of this is exponential, it contains useful notations and its ideas have been applied to algorithms which work in quasi-polynomial time [5], [9], [14]. Some FPT (fixed-parameter tractable) algorithms are known for parameters of tree-width [11], DAG-width [1], and clique-width [13]. Recently, quasi-polynomial time algorithms are designed [3], [14].

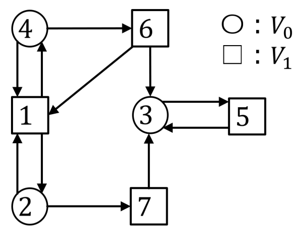

Fig. 1 Parity Game

In this paper, we introduce notions of forcing-cycles and winning-cycles, and give a polynomial-time algorithm for finding them. These ideas are useful for reducing parity games . By adapting this method, some parity games can be solved, or reduced to smaller subgames in polynomial-time.

2. Parity Game

A *parity game* $G = (V, V_0, V_1, E, \lambda)$ is a game played by two players on directed graph (V, E). Each vertex is labeled with a natural number, called *priority* $\lambda : V \to \mathbb{N}$. And the vertex set V consists of two disjoint subsets V_0 and V_1 (See **Fig. 1**). Two players P_0 and P_1 move the token along the edges of graph, starting from the given initial vertex v (we often express a game with an initial vertex as (G, v)). If the token is on a vertex in V_0, then P_0 moves the token along one of the out-going edges from the vertex; otherwise, P_1 does.

[1] Department of Computer and Network Engineering, Graduate School of Informatics and Engineering, the University of Electro-Communications

If a player cannot move the token, he/she loses the game. Otherwise, the winner depends on the maximum priority appearing infinitely in the rout that the token follows. If the priority is even, P_0 wins the game; otherwise, P_1 does. For each possible initial vertex, there is a unique player, either P_0 or P_1, who has a winning strategy. We denote by $W_0(G)$ (resp., $W_1(G)$) $\subseteq V$ the region such that if $v \in W_0(G)$ (resp., $v \in W_1(G)$) is the initial vertex, then P_0 (resp., P_1) has a winning strategy. $W_0(G)$ (resp., $W_1(G)$) may be denoted by W_0 (resp., W_1) if the game is clear from the context. If $W_0 = V$ or $W_1 = V$, then we call the game a *single-winner parity game*. Now, we define a problem *Parity Game*, whose object is to partition V into $W_0(G)$ and $W_1(G)$.

Parity Game

Input: a parity game $G = (V, V_0, V_1, E, \lambda)$.
Output: $W_0(G)$.[*1]

We also define the decision-problem version of this problem as follows. Clearly if one of them has a polynomial-time algorithm, then the other also does.

Parity Game (Decision Problem Version)

Instance: a parity game with an initial vertex $(G, v) = (V, V_0, V_1, E, \lambda, v)$.
Question: $v \in W_0(G)$?

We will often use the constants n, m, and p to mean the following values:

- $n = |V|$: the number of vertices in G.
- $m = |E|$: the number of edges in G.
- p: the maximum priority in G.

For a vertex $u \in V$, we denote by $N^+(u)$ (resp., $N^-(u)$) the set of out-neighbors (resp., in-neighbors) of u. Similarly, we define as $N^+(S) = \bigcup_{v\in S} N^+(v) \setminus S$ and $N^-(S) = \bigcup_{v\in S} N^-(v) \setminus S$. We denote by $G(S)$ the parity game on the subgraph induced by S, i.e., $G(S) = (S, V_0 \cap S, V_1 \cap S, E \cap (S \times S), \lambda_{\restriction S}$[*2]$)$. $G(V \setminus S)$ may often be written as $G \setminus S$. For a parity game G, we call a game G' a *proper subgame* of G if $G' = G(V(G'))$ and $G' \neq G$. We call each move of the token a *turn*.

For a vertex $v \in V$ and $S \subseteq V$, regardless of the strategy taken by P_1, if there exists a strategy of P_0 that can move the token placed at $v \in V$ to one of the vertices in S in i turns, then we say that P_0 *can force the token move to* S *from* v *(in* i *turns)*. If $v \in S$ or P_0 can force the token to move to S from v, then we say that P_0 *can force the token to reach* S *from* v.

We first remind an important basic property of parity games, namely that the winner of a game does not depend on previous moves; i.e., the game always admits a *memoryless (or positional) winning strategy*. This property is also called *memoryless determinacy*. A memoryless strategy for P_i is a map $f : V_i \to V$ such that $f(v)$ is an out-neighbor of v for all v such that v has an out-neighbor. When a play comes to $v \in V_i$, P_i unconditionally selects the unique out-neighbor determined by f, without the history of the play. A memoryless strategy f is a memoryles winning strategy for P_i from vertex v if every play starts from v and conforms to f is winning for P_i. When considering a memoryless strategy f of P_i, it is often useful to restrict to the graph G_f obtained from G by removing all out-going edges from vertices in V_i, except those used by f.

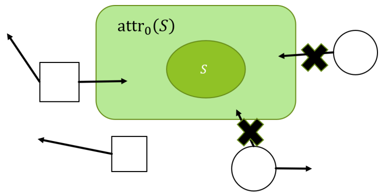

Fig. 2 0-attractor

Memoryless determinacy of parity games was proved as follows.

Corollary 1. [2] *In every parity game G, there exists memoryless strategies f_0 of P_0 and f_1 of P_1 such that P_i wins by using f_i whichever the initial vertex is in $W_i(G)$, no matter which strategy P_{1-i} applies.*

By using this corollary, the following theorem on the complexity of Parity Game DP can be proved.

Theorem 1. [6] *Parity Game DP belongs to NP $\cap$ coNP.*

For the memoryless determinacy, we only consider memoryless strategies in the following discussions.

For a parity game $G = (V, V_0, V_1, E, \lambda)$, if either $V_0 = \phi$ or $V_1 = \phi$, then it is called a *single-player parity game*. Parity Game on this game can be solved in cubic time with the number of vertices [5].

Lemma 1. *Any single-player parity game has an $O(n^3)$-time algorithm.*

3. Attractor and Dominion

In this section, we describe notations used in McNaughton's Algorithm, which was first given by McNaughton for muller games [10], and adapted to parity games by Zielonka [15]. While the running time of this algorithm is exponential, it contains useful notations: attractor, closed, and dominion.

Definition 1. [7] (See **Fig. 2**) *For $i \in \{0, 1\}$, a set of vertices U is an i-attractor of a set of vertices $S \subseteq V$, denoted by $\mathrm{attr}_i(S)$, is the smallest superset[*3] of S that satisfies the following:*

- *There are no arc from $V_i \setminus U$ to U.*
- *Every vertex of $V_{1-i} \setminus U$ has an out-neighbor outside of U.*

The i-attractor of set S is the region which P_i can force

[*1] Since $W_1 = V \setminus W_0$, W_1 is automatically obtained if W_0 is obtained.
[*2] Let $f : E \to F$ be a function from a set E to a set F. If a set A is a subset of E, then the *restriction of f to A* is the function $f_{\restriction A} : A \to F$.
[*3] The fact that this is uniquely determined can be proved from the fact that if there are two different minimal U_1 and U_2 that satisfy the condition, then $U_1 \cap U_2$ also satisfies the condition

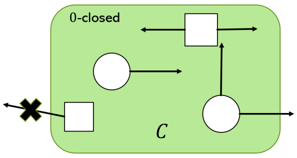

Fig. 3 0-closed

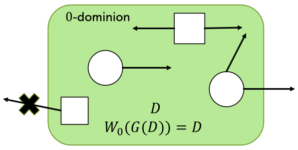

Fig. 4 0-dominion

the token to reach S from any vertex in it. We may denote $\mathrm{attr}_i(S; G)$ to specify G. An i-attractor can be calculated in polynomial-time [5], [7].

Lemma 2. *For $i \in \{0, 1\}$ and $S \subseteq V$, $\mathrm{attr}_i(S)$ can be calculated in $O(m)$-time.*

Corollary 2. *For $i \in \{0, 1\}$ and $S \subseteq V$, any vertex $v \in \mathrm{attr}_i(S) \setminus S$ satisfies the following:*
- *If $v \in V_i$, then there is an arc $(v, w) \in E$ such that $w \in \mathrm{attr}_i(S)$.*
- *If $v \in V_{1-i}$, then for every $(v, w) \in E$, we have $w \in \mathrm{attr}_i(S)$.*

Definition 2. [5] (See **Fig. 3**) *For $i \in \{0, 1\}$, $C \subseteq V$ is said to be i-closed if any $v \in C$ satisfies the following:*
(1) If $v \in V_i$, then there is some $(v, w) \in E$ such that $w \in C$.
(2) If $v \in V_{1-i}$, then for every $(v, w) \in E$, we have $w \in C$.

The fact that C is i-closed implies P_{1-i} cannot force the token to move to any vertex outside C from any vertex in C. From the definitions of i-attractors and i-closed sets, we obtain the following lemmas [7].

Lemma 3. *For $i \in \{0, 1\}$ and $S \subseteq V$, $V \setminus \mathrm{attr}_i(S)$ is $(1 - i)$-closed.*

Lemma 4. *Let $C \subseteq V$ be i-closed for $i \in \{0, 1\}$. Then $W_i(G(C)) \subseteq W_i(G)$, i.e., if P_i has a winning strategy from $v \in C$ in $G(C)$, then he/she also has a winning strategy from $v \in V$ in G.*

Definition 3. [7] (See **Fig. 4**) *For $i \in \{0, 1\}$, $D \subseteq V$ is called an i-dominion if the following hold:*
- *D is i-closed.*
- *$W_i(G(D)) = D$.*

As a clearly example, W_i is an i-dominion. But there could be smaller dominions. Attractors are useful because of making parity games more simple when we identify some parts of the region in which the winner is the same whichever the initial vertex is. The following two lemmas depicts useful relations among attractorsm and dominions, and W_i [5], [7].

Lemma 5. *Let $i \in \{0, 1\}$ and $S \subseteq V$. If $S \subseteq W_i(G)$, then $W_i = \mathrm{attr}_i(S; G) \cup W_i(G \setminus \mathrm{attr}_i(S; G))$.*

Lemma 6. *For $i \in \{0, 1\}$, let $D \subseteq V$ be an i-dominion and $S \subseteq V$ be a set of vertices such that $S \cap D = \phi$, then $\mathrm{attr}_{1-i}(S) \cap D = \phi$.*

If $D \subseteq V_i$ is i-dominion, then it can be found in polynomial-time.

Fact 1. [7] *If there exists an i-dominion D such that $D \subseteq V_i$ for some $i \in \{0, 1\}$, then it can be found in polynomial-time.*

If there exists such a dominion, then it can be removed from the graph from Lemma 5. Therefore, we assume that no player can win without passing him/her opponent vertex.

From the rules of parity games, a vertex $v \in V_{1-i}$ which has no out-going edge is clearly in W_i. Thus, the i-attractor of such a vertex is the winning region for P_i and can be removed from the original graph in polynomial-time from Lemmas 2 and 5. Therefore, in the following discussion, we assume that every vertex has at least one out-going edge.

4. Circulators, Winning-Cycles, and Parity-Game Reduction

Parity games are played on finite graphs. Therefore, if the token is moved infinitely many turns, then it contains a cycle along which the token passes infinitely. In this section, we introduce two types of cycles: forcing-cycles and winning-cycles, and present an algorithm for finding them. If we get a winning-cycle for P_i, then we find that all vertices on the cycle are in W_i. Specifically, this section consists of three subsections. Firstly, we introduce a circulator, which is the extension of an attractor. By introducing this notation, we can formally represent a set of vertices from which the player can force the token to move to certain subset of vertices. Secondly, we introduce forcing-cycles and winning-cycles, and construct a polynomial-time algorithm for finding them. Finally, we show an algorithm to reduce a parity game by removing a winning-cycle if exists. By adapting this method, some parity games can be solved, or reduced to proper subgames in polynomial-time.

4.1 Circulators

In this subsection, we introduce circulators.

Definition 4. *An i-circulator of $S \subseteq V$, for some $i \in \{0, 1\}$, denoted by $\mathrm{circ}_i(S)$, is the vertex set that is defined in the following:*

$\mathrm{circ}_i(S) = \mathrm{attr}_i(N^-(S) \cap \mathrm{attr}_i(S))$

The i-circulator of S is the region such that P_i can force the token to move to S from any vertex in it. We may denote $\mathrm{circ}_i(S; G)$ to specify G.

Fact 2. $\mathrm{attr}_i(S) = \mathrm{circ}_i(S) \cup S$.

By Lemma 3, the following holds.

Fact 3. *For $i \in \{0, 1\}$ and $S \subseteq V$, $V \setminus \mathrm{circ}_i(S)$ is $(1 - i)$-closed.*

A circulator can be calculated in polynomial-time.

Lemma 7. *For $i \in \{0, 1\}$ and $S \subseteq V$, $\mathrm{circ}_i(S)$ can be calculated in $O(m)$-time.*

Proof. The vertices of $N^-(S) \cap \mathrm{attr}_i(S)$ can be founded in $O(m)$-time by checking all edges to S. An attractor can be calculated in $O(m)$-time from Lemma 2, so the total running time is $O(m)$-time. □

Specifically, if $S \subseteq \mathrm{circ}_i(S)$, then $\mathrm{circ}_i(S) = \mathrm{attr}_i(S)$ and we obtain the following lemma

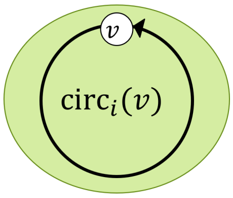

Fig. 5 Forcing-cycle

Lemma 8. *For $i \in \{0,1\}$ and $S \subseteq V$, if $S \subseteq \mathrm{circ}_i(S)$, then $\mathrm{circ}_i(S)$ is i-closed.*

Proof. Without loss of generality, assume that $i = 0$.

To simplify, we denote $A := N^-(S) \cap \mathrm{attr}_0(S)$. Then, $\mathrm{circ}_0(S) = \mathrm{attr}_0(A)$ from Definition 9. $\mathrm{attr}_0(S) = \mathrm{circ}_0(S) = \mathrm{attr}_0(A)$ from the assumption of $S \subseteq \mathrm{circ}_0(S)$ and Fact 5. Therefore, we denote by $\mathcal{A} = \mathrm{attr}_0(S) = \mathrm{circ}_0(S) = \mathrm{attr}_0(A)$. We prove that any vertex $v \in \mathcal{A}$ satisfies the following:

- If $v \in V_0$, then there is an arc $(v, w) \in E$ such that $w \in \mathcal{A}$.
- If $v \in V_1$, then for every $(v, w) \in E$, we have $w \in \mathcal{A}$.

If either $v \in \mathcal{A} \setminus S$ or $v \in \mathcal{A} \setminus A$, then from Corollary 2, v satisfies the conditions. Since $A \cap S = \phi$, every vertex in $\mathcal{A}$ satisfies the conditions. Therefore, $\mathcal{A} = \mathrm{circ}_0(S)$ is 0-closed. □

4.2 Forcing-Cycles and Winning-Cycles

Especially, if $v \in \mathrm{circ}_i(v)$ for $v \in V$, then P_i can force the token to move to v from v. We call the set of vertices u satisfying the following condition a *forcing-cycle* from v of P_i.

- Condition FC: There exists a strategy such that P_i can force the token to move to v from v and a strategy for P_{1-i} such that u is in the path from v to v according to these strategies.

The part "*from v by P_i*" is optional if it is clear from the context. By the definition, $v \in \mathrm{circ}_i(v)$ if and only if P_i has a forcing-cycle from v (See **Fig. 5**). When P_i wins by a strategy that P_i *can force the token to move to v from v*, which appears in condition FC above, the forcing-cycle corresponding to that strategy is called the *winning-cycle* from v of P_i.

Now, we give an algorithm, Winning-Cycle, for finding a winning-cycle from v of P_i, for any given vertex $v \in V$ (See Algorithm 1). In this algorithm, $(G = (V, V_0, V_1, E, \lambda), v, i)$ is the input, where $i \in \{0,1\}$. And output 1 if P_i has a winning cycle from v; otherwise 0. We show the operation in the case of $i = 0$ in the following.

Firstly, we calculate $A := \mathrm{circ}_0(v; G)$ and check if $v \in \mathrm{circ}_0(v; G)$ or not to determine whether P_0 has a forcing-cycle from v. If P_0 does not have it, then P_0 does not have a winning-cycle also. Thus, output 0 and stop. Therefore, we consider the case of that P_0 has a forcing-cycle from v. A is 0-closed from Lemma 3. Therefore, P_0 can win in the same region by using the same strategy in G in as $G(A)$. And since $V \setminus A$ is 1-closed from Fact 3, if P_0 cannot force the token to move to some vertex $z \in V \setminus A$ from some vertex $w \in A$, then P_0 cannot force to move in G as well (because P_0 cannot force the token to move to A from any vertex in $V \setminus A$). These mean that P_0's forcing-cycle from a vertices of A in G is identical to that in $G(A)$. Therefore, to determine whether P_0 has a winning-cycle from v, we need only consider $G(A)$. Let w is the vertex with maximum priority in $G(A)$.

(1) If $\lambda(w)$ is even.
We calculate $\mathrm{attr}_0(w; A)$ and check if $v \in \mathrm{attr}_0(w; A)$ or not. If $v \in \mathrm{attr}_0(w; A)$, then P_0 can force the token to move to w from v. Therefore, P_0 can force the token to move to v from v via w. Since w is the vertex with even maximum priority, P_0 has a winning-cycle from v. Thus, output 1 and stop.
Otherwise, if $v \notin \mathrm{attr}_0(w; A)$, then P_0 cannot force the token to move to w from v. In the following, we show that to determine whether P_0 has a winning-cycle from v, we need only consider the game $A \setminus \mathrm{attr}_0(w; A)$.
- If P_0 has a winning-cycle from v in $G(A \setminus \mathrm{attr}_0(w; A))$.
Consider the case of $A \setminus \mathrm{attr}_0(w; A)$ is 0-closed. Then, P_0 can take same winning strategy in $G(A \setminus \mathrm{attr}_0(w; A))$ which can force the token to move to v from v. This means P_0 has a winning-cycle.
Next, we consider the case of $A \setminus \mathrm{attr}_0(w; A)$ is not 0-closed. Then, P_1 can force the token to move to $z \in \mathrm{attr}_0(w; A)$ from some vertex in $A \setminus \mathrm{attr}_0(w; A)$. Then P_0 can force the token to move to w from z since $z \in \mathrm{attr}_0(w; A)$ and v from w, since $w \in \mathrm{circ}_0(v; G)$. Thus, if P_1 can force the token to move to z, then P_0 wins because of w is the vertex with even maximum priority. Therefore, P_0 has a winning strategy that can force the token to move to v from v regardless of strategy of P_1. This means P_0 has a winning-cycle.
- If P_0 does not have a winning-cycle from v in $G(A \setminus \mathrm{attr}_0(w; A))$.
P_0 cannot force the token to move to $\mathrm{attr}_0(w; A)$ from any vertex in $A \setminus \mathrm{attr}_0(w; A)$, because of $A \setminus \mathrm{attr}_0(w; A)$ is 1-closed from Lemma 3. $V \setminus A$ is the set of vertices of P_0 cannot force the token to move to v. Therefore, P_0 also does not have a winning-cycle in G either.

Therefore, to determine whether P_0 has a winning-cycle from v is the same result on $G(A \setminus \mathrm{attr}_0(w; A))$. So, we update A to $A \setminus \mathrm{attr}_0(w; A)$ and find w again.

(2) If $\lambda(w)$ is odd.
We calculate $\mathrm{attr}_1(w; A)$ and check if $v \in \mathrm{attr}_1(w; A)$. If $v \in \mathrm{attr}_1(w; A)$, then P_1 can force the token to move to w from v. Since w is the vertex with odd maximum priority, P_0 does not have a winning-cycle from v. Thus output 0 and stop algorithm. Otherwise, if $v \notin \mathrm{attr}_1(w; A)$, then P_1 cannot force the token to move to w from v. We can show that only $G(A \setminus \mathrm{attr}_1(w; A))$ should be considered to determine whether P_0 has a winning-cycle from v by the same argument as when $\lambda(w)$ is even. Therefore, we update A to $A \setminus \mathrm{attr}_1(w; A)$ and find w again.

Algorithm 1 WINNING-CYCLE(G, v, i)

Input: $(G = (V, V_0, V_1, E, \lambda)$, $v \in V$, $i \in \{0, 1\})$.
Output: 0 or 1.

```
if G = φ then
  return 0
end if
A := circ_i(v; G)
if v ∉ circ_i(v; G) then
  return 0
else
  while A ≠ φ do
    w := the vertex with maximum priority in A.
    j := λ(w) mod 2
    B := attr_j(w; A)
    if v ∉ B then
      A = A \ B
    else
      if i = j then
        return 1
      else
        return φ
      end if
    end if
  end while
  return φ
end if
```

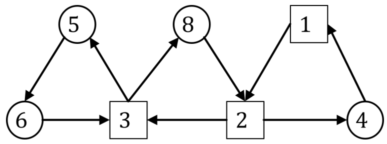

Fig. 6 Single-winner parity game without winning-cycles

Theorem 2. WINNING-CYCLE *finds a winning-cycle from v for P_i in polynomial-time.*

Proof. The correctness of the algorithm is proved by above description. Therefore, we only prove the running time of the algorithm. The running time of calculating circulator is $O(m)$-time from Lemma 7. And one step in While loop is $O(n^2)$-time, hence the running time of whole loop is $O(nm)$-time because of at least one vertex is removed in each step. Therefore, the running time of this algorithm is $O(nm)$-time. □

If P_i has a winning-cycle from v, then a larger (or equal) winning region containing the winning-cycle can be calculated in $O(m)$-time by calculating the i-attractor of v.

Note that WINNING-CYCLE can only calculate the winning region where P_i can force the token to move to v from v. Therefore, for example, in the graph of **Fig. 6**, the winning region cannot be calculated by using this algorithm. In this graph, whole graph belong to W_0, but P_0 does not have a winning-cycle from any vertex.

4.3 Parity-Game Reduction by Winning-Cycles

Now we give an algorithm WINNING-CYCLE-REDUCTION, algorithm for reducing a parity game that has winning-cycles to a proper subgame (See Algorithm 2). In this algorithm, $G = (V, V_0, V_1, E, \lambda)$ is the input. $(L_0 \subseteq W_0$, $L_1 \subseteq W_1$, $G^{'} = G \setminus (L_0 \cup L_1))$ or G is the output. We show the operation in the following.

We can check if there exists a winning-cycles in G by adapting WINNING-CYCLE for each vertex and for each player. We prepare empty sets L_0 and L_1, and if P_i has a winning-cycle from v, then we add $\mathrm{attr}_i(v; G)$ to L_i. As a result, if $L_0 \neq \phi \vee L_1 \neq \phi$, then we can calculate sets $L_0 \subseteq W_0$, $L_1 \subseteq W_1$, and a proper subgame $G^{'} = G \setminus (L_0 \cup L_1)$. Thus output them and stop. Otherwise, if $L_0 = \phi \wedge L_1 = \phi$, then G does not contain a winning-cycle. Thus, output G itself and stop.

Theorem 3. WINNING-CYCLE-REDUCTION *reduces a parity game that has winning-cycles to a proper subgame in polynomial-time.*

Proof. The correctness of the algorithm is proved by above description. Therefore, we only prove the running time of the algorithm. This algorithm applies WINNING-CYCLE to each vertex and for each i. The running time of WINNING-CYCLE is $O(nm)$-time from Theorem 4. Thus, the running time of this algorithm is $O(2n \cdot nm) = O(n^2 m)$-time. □

Specifically, if $L_0 \cup L_1 = V$, then it means G can be solved in polynomial-time.

Algorithm 2 WINNING-CYCLE-REDUCTION(G)

Input: $G = (V, V_0, V_1, E, \lambda)$.
Output: $(L_0 \subseteq W_0,\ L_1 \subseteq W_1,\ G^{'} = G \setminus (L_0 \cup L_1))$ or G.

```
L_0 := φ
L_1 := φ
for i ∈ {0, 1} do
  for v ∈ V do
    if Winning-Cycle(G, v, i) = 1 then
      L_i = L_i ∪ attr_i(v; G)
    end if
  end for
end for
if L_0 ≠ φ ∨ L_1 ≠ φ then
  return L_0, L_1, and G \ (L_0 ∪ L_1)
else
  return G
end if
```

5. Conclusion

In this paper, we introduced notions of forcing-cycles and winning-cycles. They are the vertex sets such that a player can force the token to return to the same vertex after moving the token from one vertex to another. Moreover we constructed a polynomial-time algorithm for finding them. Finally, we constructed an algorithm to reduce a parity game by removing a winning-cycle if exists. By adapting this method, some parity games can be solved, or reduced to smaller subgames in polynomial-time.

As a future task, finding the characterization of a parity game that has a winning-cycle can be mentioned. It is also important to develop an algorithm that calculates the winning region in polynomial-time even if it contains no winning-cycle.

References

[1] Berwanger, D., Dawar, A., Hunter, P. and Kreutzer, S.: DAG-width and parity games, *Annual Symposium on Theoretical Aspects of Computer Science*, Springer, pp. 524–536 (2006).
[2] Björklund, H., Sandberg, S. and Vorobyov, S.: Memoryless determinacy of parity and mean payoff games: a simple proof, *Theoretical Computer Science*, Vol. 310, No. 1-3, pp. 365–378 (2004).
[3] Calude, C. S., Jain, S., Khoussainov, B., Li, W. and Stephan, F.: Deciding parity games in quasi-polynomial time, *SIAM Journal on Computing*, No. 0, pp. STOC17–152 (2020).
[4] Chatterjee, K. and Fijalkow, N.: A reduction from parity games to simple stochastic games, *arXiv preprint arXiv:1106.1232* (2011).
[5] Dittmann, C., Kreutzer, S. and Tomescu, A. I.: Graph operations on parity games and polynomial-time algorithms, *arXiv preprint arXiv:1208.1640* (2012).
[6] Emerson, E. A., Jutla, C. S. and Sistla, A. P.: On model-checking for fragments of μ-calculus, *International Conference on Computer Aided Verification*, Springer, pp. 385–396 (1993).
[7] Gajarskỳ, J., Lampis, M., Makino, K., Mitsou, V. and Ordyniak, S.: Parameterized algorithms for parity games, *International Symposium on Mathematical Foundations of Computer Science*, Springer, pp. 336–347 (2015).
[8] Jurdziński, M.: Deciding the winner in parity games is in UP $\cap$ co-UP, *Information Processing Letters*, Vol. 68, No. 3, pp. 119–124 (1998).
[9] Jurdziński, M., Paterson, M. and Zwick, U.: A deterministic subexponential algorithm for solving parity games, *SIAM Journal on Computing*, Vol. 38, No. 4, pp. 1519–1532 (2008).
[10] McNaughton, R.: Infinite games played on finite graphs, *Annals of Pure and Applied Logic*, Vol. 65, No. 2, pp. 149–184 (1993).
[11] Obdržálek, J.: Fast mu-calculus model checking when tree-width is bounded, *International Conference on Computer Aided Verification*, Springe, pp. 80–92 (2003).
[12] Obdrzalek, J.: Algorithmic analysis of parity games, PhD Thesis, Citeseer (2006).
[13] Obdržálek, J.: Clique-width and parity games, *International Workshop on Computer Science Logic*, Springer, pp. 54–68 (2007).
[14] Parys, P.: Parity Games: Zielonka's Algorithm in Quasi-Polynomial Time, *arXiv preprint arXiv:1904.12446* (2019).
[15] Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees, *Theoretical Computer Science*, Vol. 200, No. 1-2, pp. 135–183 (1998).