# Concurrent Multi-Player Parity Games

Vadim Malvone
Università degli Studi di Napoli
Federico II, Italy
vadim.malvone@unina.it

Aniello Murano
Università degli Studi di Napoli
Federico II, Italy
murano@na.infn.it

Loredana Sorrentino
Università degli Studi di Napoli
Federico II, Italy
loredana.sorrentino@unina.it

## ABSTRACT

*Parity games* are a powerful framework widely used to address fundamental questions in computer science. In the basic setting they consist of *two-player turn-based* games, played on directed graphs, whose nodes are labeled with *priorities*. Solving such a game can be done in time exponential in the number of the priorities (and polynomial in the number of states) and it is a long-standing open question whether a polynomial-time algorithm exists. Precisely this problem resides in the class UP ∩ co-UP.

In this paper we introduce and solve efficiently *concurrent multi-player parity games* where the players, being existential and universal, compete under fixed and strict alternate coalitions. The solution we provide uses an extension of the classic *Zielonka Recursive Algorithm*. Precisely, we introduce an ad hoc algorithm for the *attractor* subroutine. Directly from this, we derive that the problem of solving such games is in PSpace. We also address the lower bound and show that the complexity of our algorithm is tight, i.e. we show that the problem is PSpace-hard by providing a reduction from the *QBF satisfiability problem*.

## Categories and Subject Descriptors

I.2.11 [**Artificial intelligence**]: Distributed artificial intelligence — *Multi-agent systems*

## General Terms

Theory, Verification

## Keywords

Parity games; Concurrent multi-player games

## 1. INTRODUCTION

*Parity games* [15, 40] are a very powerful mathematical framework widely used to address fundamental questions in computer science. They are strictly connected with other games of infinite duration and in particular with the *multi-agent* strategic reasoning [4, 8, 9, 12].

In the basic setting parity games consist of two-player turn-based games, played on directed graphs, whose nodes are labeled with *priorities* (i.e., natural numbers). The players, conventionally named $Player_0$ and $Player_1$, move in turn a token along graph's edges. Thus, a play induces an infinite path and $Player_0$ wins the play if the greatest priority visited infinitely often along the path is even; otherwise, $Player_1$ wins the play. $Player_0$ *wins* the game if he can induce a play that is winning. Otherwise $Player_1$ wins the game. The problem of finding the winner in a parity game is known to be in UPTime ∩ CoUPTime [23]. Deciding whether a polynomial-time solution exists is a long-standing open question. Aimed to solve this question, as well as to come out with solutions working efficiently in practice, several algorithms have been proposed in the last two decades. Among the others we mention the *Zielonka Recursive Algorithm* [40] as the best performing one in practice [17].

Parity games have been largely and beneficially applied in formal system design and verification [13, 14, 26, 35, 38]. In such a domain, we mainly distinguish between *closed* and *open* (reactive) systems [18, 27]. While the behavior of closed systems fully depends on internal states, open systems are characterized by an ongoing interaction with an external unpredictable environment whose interaction possibly affects the system behavior. In open system verification we check that the system satisfies a desired behavior (specification) no matter how the environment acts. In a very large setting of specification, the problem reduces to solve a parity game where the system and the environment take place as adversarial players [3, 15, 26].

In recent years, we are witnessing a strong expansion and adaptation of models and game-based reasoning in the multi-player area [1, 21, 22, 32]. In this domain, the game often involves a multitude of players who play different roles. In some simplified yet powerful settings, one can have two teams competing for opposite objectives [1]. In this scenario, if one wants to make use of parity games then there are two basic ways to follow. The first, very restricted to the specific problem under exam is to possible reduce the multi-player reasoning to a two-players one. The second, much more efficient, elegant, and general would rather introduce a multi-player parity game concept, give an efficient solution to the problem and apply it directly whenever it is required. The latter is the motivation of this paper. Our aim is indeed to come up with an abstract tool to both model reactive multi-player systems and properly solve related decision problems. In the multi-player setting these games could be efficiently applied to reasoning about the strategic interaction among the players by means of logical formalism. As a more concrete example, one can polynomially reduce formal verification

**Appears in:** *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*
Copyright © 2016, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

questions about logics for the strategic reasoning such as *Strategy Logic* [5, 10, 32, 33] and the like [29, 31] into our game setting, as well as reasoning efficiently about solution concepts in multi-player games.

In this paper we introduce and solve efficiently *Concurrent Multi-Player Parity Games* (*CMPG*, for short), in which the players behave existential or universal in strict and fixed alternation. With more details, a CMPG extends classic two-player turn-based parity games by admitting $n$ players, namely $Player_0 \dots Player_{n-1}$, to play simultaneously over the prioritized arena. Each $Player_i$, with $i \bmod 2 = 0$ acts as *existential*; the other players are instead *universal*. The aim in a CMPG is to check whether there exists a $Player_0$ strategy such that for all $Player_1$ strategies there exists a $Player_2$ strategy, and so on, such that the resulting plays satisfy the parity condition. In this case we say that the team of the existential players wins the game. Solving a CMPG amount to check whether this is the case.

We show a solution for the introduced CMPG class of games by using a non-trivial extension of the classic Zielonka Recursive Algorithm that, we recall, has been conceived for two-player turn-based parity games. Precisely, we build a new algorithm for the attractor subroutine. We show that our solution takes, as complexity, polynomial space in the size of the game. We also show that this is a tight-complexity solution. Indeed, we provide a PSPACE lower bound by using a reduction from the satisfiability problem for *Quantified Boolean Formulas* (QBF). As another evidence of the strength of our result, we show that our solution algorithm is better than a simple (direct) reduction to a two-player turn-based parity game. In fact, given a CMPG $G$, such a translation would lead to a classic parity game with a number of states that is exponential in the number of players' decision in $G$ (and thus exponential in the number of the players) but with the same number of priorities. By recalling that the Zielonka Recursive Algorithm is polynomial in the number of states but exponential in the number of priorities, we would obtain a non-tight exponential-time upper-bound.

We strongly believe that our result can be usefully used in a wide range of application domains. The verification setting we have mentioned above is surely a good example. In multi-player strategic reasoning several questions can be directly reduced to this problem. For example it can be used to address the model checking question of logics for the strategic reasoning. See [5] for an argument. It is worth mentioning that the specific setting of multi-player parity games we have introduced has never been addressed before. Indeed, all the literature on parity games mainly concentrates on solving the two-player setting [24, 25, 40], trying to reduce the time complexity [16, 17, 37] or to get parametrized algorithms [17, 19, 28, 36], as well as to apply them in the synthesis and verification of reactive systems [2, 27, 28]. In case a multi-player system is faced, then parity game have been always applied by first introducing a two-player intermediate setting [1, 11]. Finally, we report that concurrent two-player parity games have been considered in the stochastic setting [6, 7, 12, 39].

**Outline** The sequel of the paper is structured as follows. In Section 2, we introduce Concurrent Multi-Player Parity Structures and other preliminary concepts about games. In Section 3, we report the classic Zielonka Recursive Algorithm used to solve turn-based parity games. In Section 4, we introduce CMPG and provide a first simple but non-tight solution via a translation to turn-based two-player parity games. In Section 5 we provide a PSPACE solution algorithm to solve CMPG by introducing a new appropriate Attractor function of the Zielonka Recursive Algorithm to work with multiple players. In Section 6, we study the lower bound of the address problem and show that it is PSPACE-HARD by means of a reduction from the QBF satisfiability problem. Finally, in Section 7 we give some conclusions.

## 2. MODEL

In this section, we introduce a semantic framework, namely a *Concurrent Multi-Player Parity Structure* as an abstract game arena in which a number of players, concurrently and independently, perform their moves. Such a structure is obtained by further equipping classic *Concurrent Game Structures* [1, 32] with a labeling function that assigns to each state a natural number called *priority*. The definition follows.

DEFINITION 2.1. *A* Concurrent Multi-Player Parity Structure *(*CMPS*) is a tuple* $\mathcal{G} \triangleq < \mathrm{Pl}, \mathrm{Ac}, \mathrm{St}, s_I, \lambda, \mathsf{tr} >$*, where* $\mathrm{Pl} = \{Player_0, \dots, Player_{|\mathrm{Pl}-1|}\}$ *is a finite non-empty set of* players*,* Ac *and* St *are enumerable non-empty sets of* actions *and* states*,* $s_I \in \mathrm{St}$ *is a designated* initial state*, and* $\lambda : \mathrm{St} \to \mathrm{N}$ *is a* labeling function *that assigns to each state a natural number named* priority*. Let* $\mathrm{Dc} \triangleq \mathrm{Pl} \rightharpoonup \mathrm{Ac}$ *be the set of* decisions*, i.e., partial functions describing players' action choices. Also, let* $d[i \to a]$ *be the decision that differs from* $d$ *on the fact that* $Player_i$ *chooses action* $a$*. Then,* $\mathsf{tr} : \mathrm{Dc} \to (\mathrm{St} \rightharpoonup \mathrm{St})$ *denotes the* transition function *mapping every decision* $\delta \in \mathrm{Dc}$ *to a partial function* $\mathsf{tr}(\delta) \subseteq \mathrm{St} \times \mathrm{St}$ *representing a deterministic graph over the states.*

We will use CMPS to define the semantics of Concurrent Multi-Player Parity Games (CMPG). Before doing that we need some additional basic game concepts.

A *track* is a finite sequence of states $\rho \in \mathrm{St}^*$ such that, for each $i \in [0, \cdots, |\rho| - 1[$, there exists a decision $d \in \mathrm{Dc}$ such that $(\rho)_{i+1} = \mathsf{tr}((\rho)_i, d)$. A track $\rho$ is *non trivial* if $|\rho| > 0$. We use $\mathrm{Trk} \subseteq \mathrm{St}^+$ to denote the set of all non trivial traces. Moreover, by $\mathsf{fst}(\rho) = \rho_0$ we denote the first state of a track. By $\mathrm{Trk}(s) \subseteq \mathrm{Trk}$ we denote the set of tracks starting from the state $s$, *i.e.* $\mathrm{Trk}(s) \triangleq \{\rho \in \mathrm{Trk} : \mathsf{fst}(\rho) = s\}$.

A *path* $\pi$ is an infinite sequence of states $\pi \in \mathrm{St}^\omega$ such that, for each $i \in \mathrm{N}$, there exists a decision $d \in \mathrm{Dc}$ such that $(\pi)_{i+1} = \mathsf{tr}((\pi)_i, d)$. We use $\mathrm{Pth} \subseteq \mathrm{St}^\omega$ to indicate the set of all paths and $\mathrm{Pth}(s) \subseteq \mathrm{Pth}$ for the set of paths starting from the vertex $s$, that is $\mathrm{Pth}(s) \triangleq \{\pi \in \mathrm{Pth} : \mathsf{fst}(()\pi) = s\}$.

A CMPG is a game played over a CMPS by players $Player_0, \dots, Player_{|\mathrm{Pl}-1|}$ under the assumption that they will play for an infinite number of rounds. In each round, the players concurrently and independently choose moves, and the current state and the action chosen for each player determine the successor state. In details we have that each $Player_i$, with $i \bmod 2 = 0$ is part of the *existential (even) team*; the other players are instead part of the *universal (odd) team*. Informally, the goal in a CMPG is to check whether there exists a $Player_0$ strategy such that for each $Player_1$ strategy there exists a $Player_2$ strategy, and so forth, such that the induced plays satisfy the parity condition. Then, we say that the existential player team *wins* the game. A *strategy* can be seen as a scheme that, for each player, contains all the choices of actions he can perform depending on the track of the game. The formal definition follows.

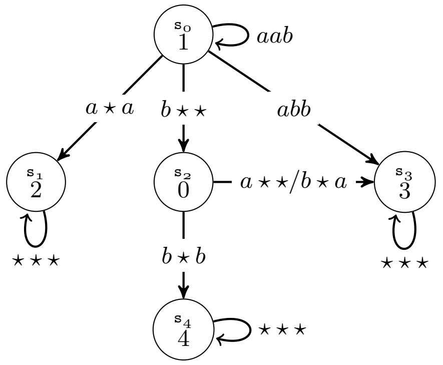

**Figure 1: Concurrent Multi-Player Parity Game.**

Definition 2.2. *A* strategy *is a partial function $\sigma : \mathrm{Trk} \rightharpoonup \mathrm{Ac}$ that maps each non-trivial internal track in its domain to an action. Given a state $s \in \mathrm{St}$, a strategy $\sigma$ is said $s$*-total *if it is defined on all tracks starting in $s$, i.e., $\mathsf{dom}(\sigma) = \mathrm{Trk}(s)$. $\mathrm{Str}$ (resp., $\mathrm{Str}(s)$) denotes the set of all (resp., $s$-total) strategies. Additionally, for a player $Player_i \in \mathrm{Pl}$, we indicate with $\mathrm{Str}_i(s)$ the set of all strategies for $Player_i$ starting from $s$ and with $\sigma_i \in \mathrm{Str}_i(s)$ one of its strategies.*

A *play* is an outcome of the game determined by the strategies of all the players being involved. Its definition follows.

Definition 2.3. *Given a fixed strategy $\sigma_i \in \mathrm{Str}_i(s)$ for each $Player_i \in \mathrm{Pl}$, the composition of these strategies induces a unique path $\pi = \pi_0\pi_1\pi_2\ldots$ named* play *such that for all $j \in \mathrm{N}$, it holds that $(\pi)_{j+1} = \mathsf{tr}((\pi)_j, d_j)$, where $d_j$ is the decision $\sigma_0((\pi)_{\leq j}) \times \ldots \times \sigma_{|\mathrm{Pl}-1|}((\pi)_{\leq j})$.*

For the sake of clarity we introduce the following example.

Example 2.1. *Consider the CMPG depicted in Figure 1. It models the interaction between three players $\mathrm{Pl}=\{Player_0, Player_1, Player_2\}$ over an arena with five states, $\mathrm{St} = \{s_0, s_1, s_2, s_3, s_4\}$, where $s_0$ is the initial state. Each state of the arena is labeled with a natural number, reported under the name of the graph vertex, as defined by the priority function: $\lambda(s_0) = 1$, $\lambda(s_1) = 2$, $\lambda(s_2) = 0$, $\lambda(s_3) = 3$, $\lambda(s_4) = 4$. The transition function is so defined: $\mathsf{tr}(s_0, a \star a) = s_1$, $\mathsf{tr}(s_0, b \star\star) = s_2$, $\mathsf{tr}(s_0, abb) = s_3$, $\mathsf{tr}(s_0, aab) = s_0$ $\mathsf{tr}(s_1, \star\star\star) = s_1$, $\mathsf{tr}(s_2, a \star\star) = s_3$, $\mathsf{tr}(s_2, b \star a) = s_3$, $\mathsf{tr}(s_2, b \star b) = s_4$, $\mathsf{tr}(s_3, \star\star\star) = s_3$ and $\mathsf{tr}(s_4, \star\star\star) = s_4$, for all $\star \in \mathrm{Ac}$. A possible play is $\pi = s_0 s_2 {s_4}^{\omega}$. This play is obtained by letting the players to take in traces $(s_0)$ $(s_0s_2)$ $(s_0s_2s_4)^{\omega}$ the following actions, respectively: for $Player_0$ the actions $b$, $b$, $\star$, for $Player_1$ the actions $\star\star\star$, and for $Player_2$ the actions $\star$ $b$ $\star$.*

## 3. ZIELONKA RECURSIVE ALGORITHM

A *Turn-Based Two-Player Parity Game* (T2PG, for short) is played on a Turn-Based Two-Player Game Structure (T2PS), an arena in which the states are partitioned into two sets and respectively assigned to the two players $Player_0$ and $Player_1$. These players move a (single and shared) token along the edges of the graph and a player moves the token to an adjacent node when he owns it along one of its states. As in CMPG, the moves of the player induce an infinite play and $Player_0$ wins this play if the largest priority that occurs infinitely often is even; otherwise $Player_1$ wins. Formally, the winner of the play $\pi = s_0, s_1, ... \in \mathrm{Pth}$ under the *parity condition* is $Player_i$ iff $\mathsf{max}\{p : \forall j\ \exists k \geqslant j : \lambda(s_k) = p\} \mod 2 = i$. $Player_0$ wins the game if he is can prevent $Player_1$ to induce a play in which $Player_1$ is the winner.

It is useful to observe that, if we consider a CMPG with two players in which in each state only one player can choose an action (this is possible as we have set *decision function* to possibly be partial along the definition) then we obtain a T2PG. Therefore, in the following we assume a T2PG to be just a CMPG with two players and equipped with a specific partial decision function, as described before, that effectively let partitioning the set of states among the two players.

In the literature, several algorithms have been introduced to solve T2PG and, among the others, the *Zielonka Recursive Algorithm* [40] has been shown to perform better than the others in randomly generated games [17]. This slgorithm, as reported in Figure 2, uses a divide and conquer technique. It constructs the winning sets for both players using the solution of sub-games. It removes the nodes with the highest priority from the game, together with all nodes (and edges) attracted to this set. The algorithm $Solve(G)$ takes as input a game graph $G$ and, after a number of recursive calls over ad hoc built sub-games, returns the winning sets $\mathrm{W}_0$ and $\mathrm{W}_1$ for $Player_0$ and $Player_1$, respectively. It constructs the winning sets $\mathrm{W}_0$ and $\mathrm{W}_1$ for both players using the solution of sub-games. In particular, it makes use of a subroutine called *attractor*. Informally, starting from a set $X \subseteq \mathrm{St}$ and $Player_i$, with $i \in \{0, 1\}$, the $i$-attractor of $X$ is the least set of nodes $Attr_i(X)$ containing $X$ such that $i$ can force a visit to $X$ from every node in $Attr_i(X)$. In other words, it starts with the initial set $X$ and adds, in every step, all nodes belonging to $Player_i$ that can reach $X$ with a single edge and all nodes belonging $Player_{1-i}$ that must reach $X$ no matter which edge $Player_{1-i}$ takes. Since calculating the reachable vertexes from a set of vertexes $X$ requires at most time $O(|e| + |n|)$, we have that, in the worst case, calculating $Attr_i(X)$ requires time $O(|n|^2)$.

```
Algorithm $Solve(G)$:
if $\mathrm{St} = \emptyset$ then
 $\mathrm{W}_0 = \emptyset$;
 $\mathrm{W}_1 = \emptyset$;
if $\mathrm{St} \neq \emptyset$ then
 $p = \mathsf{max}\{\lambda(s) | s \in \mathrm{St}\}$;
 $i = p \mod 2$;
 $X = \{s \in \mathrm{St} : \lambda(s) = p\}$;
 $A = Attr_i(G, X)$;
 $(\mathrm{W}'_i, \mathrm{W}'_{\neg i}) = Solve(G \setminus A)$;
  if $\mathrm{W}'_{\neg i} = \emptyset$ then
   $\mathrm{W}_i = A \cup \mathrm{W}'_i$;
   $\mathrm{W}_{\neg i} = \emptyset$;
  else
   $B = Attr_{\neg i}(G, \mathrm{W}'_{\neg i})$;
   $(\mathrm{W}''_i, \mathrm{W}''_{\neg i}) = Solve(G \setminus B)$;
   $\mathrm{W}_i = \mathrm{W}''_i$;
   $\mathrm{W}_{\neg i} = B \cup \mathrm{W}''_{\neg i}$;
return $(\mathrm{W}_i, \mathrm{W}_{\neg i})$
```

**Figure 2: Zielonka Recursive Algorithm.**

The following theorem reports the complexity of the Zielonka Recursive Algorithm.

Theorem 3.1. *[40] For a* T2PG $\mathcal{G}$ *with n states, e edges, and k priorities, the algorithm Solve(G), returning the winning sets for both players, takes time complexity* $O(e \cdot n^k)$.

## 4. SOLVING CMPG VIA A REDUCTION TO T2PG

In this section, we introduce an algorithm to solve CMPG through a direct reduction to T2PG. As we will see later, this gives a non-efficient solution. Indeed, in Section 5 we show a better performing algorithm resulting from a non-trivial extension of the Zielonka Recursive Algorithm. Precisely, we introduce an ad hoc algorithm for the attractor subroutine.

In the reduction we propose from a CMPG to a T2PG, the players are not treated in a one-shot fashion, but rather they are emulated by finite games between the two players: the first chooses the actions of the players of the existential team and the second the ones from the universal one. The resulting T2PG is just polynomial in the number of actions and exponential in the number of players of the starting CMPG.

Theorem 4.1. *Given a* CMPG $\mathcal{G}$ *with* $|\mathrm{St}|$ *states and k priorities, there is a* T2PG $\mathcal{G}^\star$, *played by* $Player_0$ *and* $Player_1$ *over a* T2PS *with* $|\mathrm{St}|\cdot|\mathrm{Pl}|\cdot|\mathrm{Ac}^{\mathrm{Pl}}|$ *states and* $O(|k|)$ *priorities such that* $Player_0$ *wins the* T2PG $\mathcal{G}^\star$ *iff the existential player team wins the* CMPG $\mathcal{G}$.

Proof (Sketch). Let $\mathcal{G} \triangleq < \mathrm{Pl}, \mathrm{Ac}, \mathrm{St}, s_I, \lambda, \mathsf{tr} >$ be a CMPG, we now show how to construct the required T2PG $\mathcal{G}^\star \triangleq < \mathrm{Pl}^\star, \mathrm{Ac}^\star, \mathrm{St}^\star, s_I{}^\star, \lambda^\star, \mathsf{tr}^\star >$.

By definition, $\mathcal{G}^\star$ has two players, $Player_0$ and $Player_1$, and $Player_0$ wins the game if he is able to induces only plays in which the greatest priority that occurs infinitely often along the corresponding paths in the structure of the game is even; otherwise $Player_1$ wins the game. To achieve this task, for each state in St, $Player_0$ (resp., $Player_1$) chooses an appropriate action for each player even (resp., odd). Thus we set $\mathrm{Pl}^\star \triangleq \{Player_0, Player_1\}$ and $\mathrm{Ac}^\star \triangleq \mathrm{Ac}$. The state space has to maintain an information about the position in $\mathcal{G}$ together with the index of the player that has still to be evaluated and the actions already associated to the previous players. To do this, we set $\mathrm{St}^\star \triangleq \mathrm{St} \times [0, |\mathrm{Pl}| - 1] \times_i (Dc_{<i})$ where $\mathrm{Dc}_{<i}$ is the decision of $Player_j$, for each $j = 0, \ldots, i-1$.

Before proceeding with the definition of the transition function, it is helpful to identify which are the active players for each possible state. We define a function $\mathsf{type} : \{0, \ldots, |\mathrm{Pl}|\} \to \mathrm{Pl}^\star$ that for an index of player gives the corresponding player in $\mathcal{G}^\star$. For a state $(s, i, d) \in \mathrm{St}^\star$ the active player is $\mathsf{type}(i) \triangleq i \; mod \; 2$.

The transition function is defined as follows. For each state $(s, i, d)$ with $i < |\mathrm{Pl}| - 1$ and decision $\mathsf{type}(i) \mapsto c$, we simply need to increase the counter $i$ and associate the player $i$ with action $c$. Formally, we set $\mathsf{tr}^\star((s, i, d)) \triangleq (s, i+1, d[i \mapsto c])$. For a state $(s, |\mathrm{Pl}| - 1, d)$, instead, we define a transition to the state $(s', 0, \varnothing)$, where $s'$ is the successor of $s$ in the game $\mathcal{G}$ following the decision $d[|\mathrm{Pl}| - 1 \mapsto c]$, whenever active. Formally, we have $\mathsf{tr}^\star((s, |\mathrm{Pl}| - 1, d)) \triangleq (\mathsf{tr}(s, d[|\mathrm{Pl}| - 1 \mapsto c]), 0, \emptyset)$. Now, we define the priority function $\lambda^\star$. For each state $(s, i, d)$, we have that:

$$\lambda^\star((s, i, d)) \triangleq \begin{cases} \lambda(s) + 2, & \text{if } i = 0 \text{ and } d = \emptyset; \\ 0, & \text{otherwise.} \end{cases}$$

The idea behind the priority function is to simulate in an equivalent manner the concurrent game. Given a transition from $s$ to $s'$ in $\mathcal{G}$ with $d$ the actions associated to each player, the effective transition in $\mathcal{G}$ occurs when the game is in $(s, |\mathrm{Pl}| - 1, d')$ with $d'$ in accordance with $d$, and the player $|\mathrm{Pl}| - 1$ selects the action in accordance with $d$, then the game passes at the state $(s', 0, \emptyset)$. We name all states $(s, i, d)$ with $i = 0$ and $d = \emptyset$ the external states and internal the others. Our goal is to use a small priority for the internal states, *i.e.* the states that do not correspond to any state in the original game. To do this, we label all internal states with the lowest priority (*i.e.* 0) and all external states by increasing their priority by a factor of 2. We increase the priority of external states to avoid the case that the internal states may have the same priority of the external once. Finally, the initial state is $s_I^\star \triangleq (s_I, 0, \varnothing)$.

Now, we discuss the complexity of reduction from $\mathcal{G}$ to $\mathcal{G}^\star$. First, we have that for each $s \in \mathrm{St}$ the reduction produces a number of states equal to $\prod_{|\mathrm{Pl}|} |\mathrm{Ac}^{\mathrm{Pl}}|$. In fact, given a state $s$ for each player, in the worst case, the number of states produced by the reduction is $|\mathrm{Ac}^{\mathrm{Pl}}|$, *i.e.* the cardinality of the set of decisions. Then, the complexity result follows. □

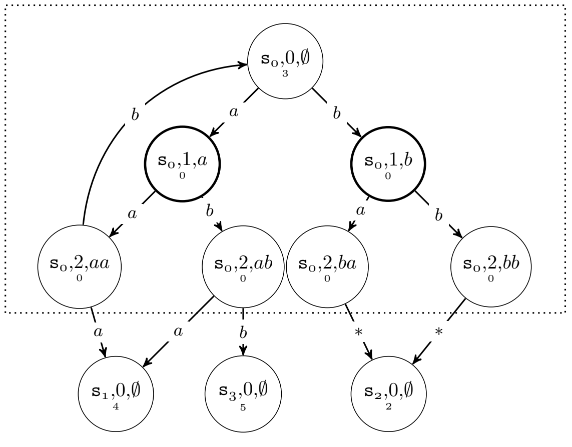

**Figure 3: Conversion of the initial state.**

To better understand the reduction in the previous theorem, consider again the model introduced in the previous section and in particular its initial state $s_0$. In gray rectangle depicted in Figure 3 there are the states that represent $s_0$ in a two-player turn-based parity game. In particular, there are three level of decision, two for $Player_0$ and one for $Player_1$ (depicted in bold). After the three level of decision, as in Figure 1 there are the effective transactions. For example, consider the state $(\mathsf{s}_o, 2, aa)$. The owner of this state is $Player_0$ in fact $\mathsf{type}(2) = 0$. If $Player_0$ chooses the action $b$ it returns to state $(\mathsf{s}_o, 0, \emptyset)$ that in $\mathcal{G}$ is represented through a loop on initial state $s_0$.

By the construction showed in the Theorem 4.1 and by the complexity for solving a T2PG reported in Theorem 3.1, we obtain the following result.

Corollary 4.1. *The complexity to solve the* CMPG *is exponential in both the number of the priorities and the number of players.*

## 5. SOLVING EFFICIENTLY A CMPG

The main complexity bottleneck in dealing with CMPG is the number of players. Indeed, the attractor procedure needs to follow the full chain of players to come up with the winning set of states and this requires, in the worst case, exponential time. In this section we provide some tools in order to simplify this task and then we show how to apply them to have an efficient solution algorithm to solve the game.

Generally speaking, standard quantification over strategies introduces an unintuitive interpretation in the semantics of games, since it instantiates full strategies, which includes also future actions, whereas players see only actions played in the past. In more details, a quantification of the form $\forall\exists$ means that the existentially quantified strategy is chosen in response to the universally quantified strategy, which prescribes actions in response to tracks that may never arise in the actual game. In classic games, a player reacts only to current and past actions taken by other players, so the response of the existentially quantified strategy to a given track should depend only on the response of the universally quantified strategy to this track. We deeply make use of this concept along the proposed solution for CMPG. To explain this formally, we make use of the concept of *Skolem's dependence function* of a quantification, as introduced in [32]. The main idea here is inspired by what Skolem proposed for *First Order Logic* in order to eliminate all existential quantifications over variables, by substituting them with second order existential quantifications over functions, whose choice is uniform *w.r.t.* the universal variables.

We start by introducing some notation regarding quantification prefixes.

Let $\wp \in Qnt(\mathrm{V})$ be a quantification prefix over a set $V(\wp) \triangleq \mathrm{V} \subseteq \mathrm{Vr}$ of variables. By $\langle\langle\wp\rangle\rangle \triangleq \{x \in V(\wp) : \exists i \in [0, |\wp|[ .\ (\wp)_i = \langle\langle x\rangle\rangle\}$ and $[[\wp]] \triangleq V(\wp) \setminus \langle\langle\wp\rangle\rangle$ we denote the sets of *existential* and *universal variables* quantified in $\wp$, respectively.

Let $x, y \in V(\wp)$ be two variables. We say that $x$ *precedes* $y$ in $\wp$, in symbols $x <_\wp y$, if $x$ occurs before $y$ in $\wp$, i.e., there are two indexes $i, j \in [0, |\wp|[$, with $i < j$, such that $(\wp)_i \in \{\langle\langle x\rangle\rangle, [[x]]\}$ and $(\wp)_j \in \{\langle\langle y\rangle\rangle, [[y]]\}$. We also say that $y$ is *functional dependent* on $x$, in symbols $x \rightsquigarrow_\wp y$, if $x \in [[\wp]]$, $y \in \langle\langle\wp\rangle\rangle$, and $x <_\wp y$, i.e., $y$ is existentially quantified after that $x$ is universally quantified, so, there may be a dependence between a value chosen by $x$ and that chosen by $y$. This definition induces the set $Dep(\wp) \triangleq \{(x, y) \in V(\wp) \times V(\wp) : x \rightsquigarrow_\wp y\}$ of *dependence pairs* and its derived version $Dep(\wp, y) \triangleq \{x \in V(\wp) : x \rightsquigarrow_\wp y\}$ containing all variables from which $y$ depends.

We use $\overline{\wp} \in Qnt(V(\wp))$ to indicate the quantification derived from $\wp$ by *dualizing* each quantifier contained in it, i.e., for all indexes $i \in [0, |\wp|[$, it holds that $(\overline{\wp})_i = \langle\langle x\rangle\rangle$ iff $(\wp)_i = [[x]]$, with $x \in V(\wp)$. It is evident that $\langle\langle\overline{\wp}\rangle\rangle = [[\wp]]$ and $[[\overline{\wp}]] = \langle\langle\wp\rangle\rangle$.

Finally, we define the notion of *valuation* of variables. Let D a generic set, called *domain*, by $Val_{\mathrm{D}}(\mathrm{V}) \triangleq \mathrm{V} \to \mathrm{D}$ we denote the set of all valuation functions over D defined on $\mathrm{V} \subseteq \mathrm{Vr}$. In the sequel of the paper we consider as domain set just the set of actions.

We now give a general high-level semantics for the quantification prefixes by means of the following definition of *Skolem dependence function*.

Definition 5.1 (Skolem Dependence Function). *Let $\wp \in Qnt(\mathrm{V})$ be a quantification prefix over a set $\mathrm{V} \subseteq \mathrm{Vr}$ of variables, and D a set. Then, a* Skolem dependence function *for $\wp$ over D is a function $\theta : Val_{\mathrm{D}}([[\wp]]) \to Val_{\mathrm{D}}(\mathrm{V})$ satisfying the following properties:*

1. *$\theta(v)_{\restriction [[\wp]]} = v$, for all $v \in Val_{\mathrm{D}}([[\wp]])$;*
2. *$\theta(v_1)(x) = \theta(v_2)(x)$, for all $v_1, v_2 \in Val_{\mathrm{D}}([[\wp]])$ and $x \in \langle\langle\wp\rangle\rangle$ such that $v_{1\restriction Dep(\wp,x)} = v_{2\restriction Dep(\wp,x)}$.*

*By $\Theta_{\mathrm{D}}(\wp)$ we denote the set of all Skolem dependence functions for $\wp$ over D. It is important to note that, by replacing D with the set of strategies, we obtain a Skolem dependence function over strategies.*

We have now all the ingredients to give an appropriate definition of the Attractor function for a given CMPG $G$. Recall that, differently from the turn-based case, we do not have the specific membership of a state to a specific player. Also, we do not have two single players but rather two teams of players, one existential and one universal, with the players of the two teams playing in a precise shuffled modality. Accordingly, we introduce two different sub-notions of Attractor, one used for the team of players to which is associated the existential quantifier $Attr^{\exists}(G, X)$ and the other for the one to which it is associated the universal quantifier $Attr^{\forall}(G, X)$. The two definitions follow.

Definition 5.2. *$Attr^{\exists}(G, X)$ is the largest set of vertexes such that the players associated to the existential quantifier have a strategy that allows to move from any vertex of $Attr^{\exists}(G, X)$ to $X$ in a finite number of steps. Formally, for each $k \in \mathrm{N}$ we define*
*$X_0 = X$;*
*$X_{k+1} = X_k \cup \{s \in \mathrm{St} : \exists\theta \in \Theta_{\mathrm{Ac}}(\wp), \forall c \in Val_{\mathrm{Ac}}([[\wp]]) : tr(s, \theta(c)) \in X_k\}$.*
*$Attr^{\exists}(G, X) = \bigcup_{k \in \mathrm{N}} X_k$.*

Definition 5.3. *$Attr^{\forall}(G, X)$ is the largest set of vertexes such that the players associated to the universal quantifier have a strategy that allows to move from any vertex of $Attr^{\forall}(G, X)$ to $X$ in a finite number of steps. Formally, for each $k \in \mathrm{N}$ we define*
*$X_0 = X$;*
*$X_{k+1} = X_k \cup \{s \in \mathrm{St} : \exists\theta \in \Theta_{\mathrm{Ac}}(\neg\wp), \forall c \in Val_{\mathrm{Ac}}([[\wp]]) : tr(s, \theta(c)) \in X_k\}$.*
*$Attr^{\forall}(G, X) = \bigcup_{k \in \mathrm{N}} X_k$.*

Similarly, we can define the sets of vertexes from which the existential $\exists$ or the universal $\forall$ team gets trapped, so losing the game. The formal definition of the trap notion follows.

Definition 5.4. *Let $i \in \{\exists, \forall\}$. A $\sigma_i$-*trap *over $G$ is any non-empty set $X$ of vertexes such that for each vertex $s \in X$, if $i = \exists$ there exists a function $\theta \in \Theta_{\mathrm{Ac}}(\neg\wp)\forall c \in Val_{\mathrm{Ac}}([[\wp]])$ such that $tr(s, \theta(c)) \in X$, otherwise there exists a function $\theta \in \Theta_{\mathrm{Ac}}(\wp)\forall c \in Val_{\mathrm{Ac}}([[\wp]])$ such that $tr(s, \theta(c)) \in X$.*

Below, we report some basic properties about the construction of the winning sets for the coalitions of players $\exists$ and $\forall$. The following facts are immediate remarks on the definition of the game and extend to our game setting similar concepts from the classic two-player turned-based parity games.

```
1  Algorithm Attr(G,X,f,n):
2  c = $|X|$;
3  for k = 1 to c do
4   v = $X_k$;
5   h = $numadj(X_k) \backslash removeAdjIn(v, X)$;
6   while $h \neq 0$
7    u = removeFirst(adj(v));
8    h = h−1;
9    if $Check(n, f, 0, \emptyset, s, X) = 1$ then
10    $X = X \cup u$;
11 return (X)
```

**Figure 4: Multi-player Attractor Algorithm.**

```
1  Algorithm Check(n,f,i,d, s, X):
2  r = $i$ mod 2;
3  for $c \in \mathrm{Ac}$ do
4   if $i = n - 1$ then
5    if $i$ mod $2 = f$
6     $r = (r \vee \mathsf{tr}(s, d[i \rightarrow c]) \in X)$;
7     else
8     $r = (r \wedge \mathsf{tr}(s, d[i \rightarrow c]) \in X)$;
9   else
10   if $i$ mod $2 = f$
11    $r = (r \vee Check(n, f, i + 1, d[i \rightarrow c], s, X))$;
12    else
13    $r = (r \wedge Check(n, f, i + 1, d[i \rightarrow c], s, X))$;
14 return (r)
```

**Figure 5: Check Algorithm.**

Remark 5.1. *Let $G$ be a* CMPG*. For each set $X \subseteq \mathrm{St}$ we have that $\mathrm{St} \setminus Attr^{\forall}(G, X)$ is a trap for the players associated to the universal quantifier.*

Remark 5.2. *Let $G$ be a* CMPG*. For each set $X \subseteq \mathrm{St}$ we have that $\mathrm{St} \setminus Attr^{\exists}(G, X)$ is a trap for the players associated to the existential quantifier.*

Finally, the next remark refers to the winning sets $W$ constructed in the algorithm $Solve(G)$ in Figure 2.

Remark 5.3. *For $i \in \{\exists, \forall\}$ and $X \subseteq \mathrm{St}$. If $\mathrm{W}'_{\neg i} = \emptyset$ then $\mathrm{W}_i = Attr^i(G, X) \cup \mathrm{W}'_i$ is a winning set for a set of player $i$.*

In the sequel we describe how to compute the attractor sets. We also discuss the time and space complexity of the related routines.

Theorem 5.1. *Let $G$ be a* CMPG*, $p \in \{\exists, \forall\}$, and $X \subseteq Attr^p(G, X)$. Let $m = |\mathrm{St}|$ be the number of vertexes in $G$. The set $Attr_p(G, X)$ is computable in time $2^{O(log(m)+log|\mathrm{Ac}|*n)}$ where $n$ is the number of players and* Ac *the set of actions.*

Proof. Consider the algorithm $Attr(G, X, f, n)$ in Figure 4. It takes as input a game graph $G$, a set $X \subseteq \mathrm{St}$ of vertexes, a flag $f$ indicating whether it is the turn of the player belonging to the team even or odd, and $n$ corresponding to the number of players. Suppose that the vertexes are ordered. First, the algorithm computes the size of the set $X$ ($c = |X|$). For each vertex in $X$, its adjacency list is scanned, checking if every adjacent vertex satisfies the IF condition in line 9. The algorithm terminates when all the adjacency lists of the vertexes in $X$ have been exploited. Then the algorithm returns the set $X$ containing the largest set of vertexes form which, in a finite number of steps, it is able to reach a vertex in $X$. The algorithm in the mentioned IF condition makes a call to the subroutine $Check(n, f, 0, \emptyset, s, X)$. This function takes as input $n$ corresponding to the number of players, a flag $f$ indicating whether it is the turn of a player belonging to the team even or odd, an index $i$ tracking the player who is playing at the current time, a decision $d$ representing the choice of an action for each player, a vertex $s$ and a set $X$ that are used to determine whether, from $s$ and performing the decision $d$, it is possible to stay in $X$. It assigns, from a vertex $s$ taken as input, an action to a player and then checks whether this choice allows to achieve the target. Let analyze the time complexity of the subroutine $Check(n, f, 0, \emptyset, s, X)$. We indicate this value with $T_{Check(n)}$. This complexity clearly relies on the construct FOR in line 3 that gives the major contribution over the input parameters. This construct is repeated at most $|\mathrm{Ac}|$ times. To calculate $T_{Check(n)}$, we calculate the complexity of a generic $T_{Check(n-i)}$. Let $T_{Check(n-i)}$ be a positive function expressed as follows.

$$T_{Check(n-i)} = \begin{cases} 2 + 3 * |\mathrm{Ac}| & \text{if } i = n - 1 \\ 2 + (3 + T_{Check(n-(i+1))}) * |\mathrm{Ac}| & \text{if } i < n - 1 \end{cases}$$

By replacing, iteratively, the values of the variable $i$, we obtain $T_{Check(n)} = 2 + 5 * \sum_{i=1}^{n-1} |\mathrm{Ac}|^i + 3 * |\mathrm{Ac}|^n$. From this we can deduce what follows.

$$\begin{aligned} T_{Check(n)} &= 2 + 5 * \textstyle\sum_{i=1}^{n-1} |\mathrm{Ac}|^i + 3 * |\mathrm{Ac}|^n \\ &\leq 2 + 5 * (n-1) * |\mathrm{Ac}|^{n-1} + 3 * |\mathrm{Ac}|^{n\ 1} \\ &\leq (n-1) * |\mathrm{Ac}|^{n-1} \\ &= O(n * (|\mathrm{Ac}|^n \div |\mathrm{Ac}|)) \\ &= O(n * |\mathrm{Ac}|^{n-1}) \\ &= O(n * |\mathrm{Ac}|^n) \\ &= O(n * 2^{\log |\mathrm{Ac}| * n}) \\ &= O(2^{\log n} * 2^{\log |\mathrm{Ac}| * n}) \\ &= O(2^{\log n + \log |\mathrm{Ac}| * n}) \\ &= 2^{O(\log |\mathrm{Ac}| * n)} \end{aligned}$$

So far, we have calculated the complexity of the subroutine $Check(n, f, 0, \emptyset, s, X)$.

Now, we analyze the complexity of the attractor function $Attr(G, X, f, n)$ . The assignment at the line 1 is done in constant time. The FOR (line 3...11) is performed $|X|$ times that, in the worst case, is equal to $|\mathrm{St}|$. The WHILE construct (line 6) is performed $h$ times. Based on these considerations, we say that the running time of the attractor function is given by $\sum_{k=1}^{|X|} 3 + (4 + T_{Check}(n))h_k$.

Assume that $m = |\mathrm{St}|$ and $j = |X|$. It is easy to see that $h_k = m - j$ (line 5). So, we have that $\sum_{k=1}^{j} 3 + (4 + T_{Check}(n))k(m - k)$. By a matter of calculation and over approximation, we get that the time complexity of the attractor is $2^{O(log(m)+log|\mathrm{Ac}|*n)}$. □

[1] Note that $|\mathrm{Ac}| > (n - 1)$

It is clear that the extra time complexity in solving a CMPG, with respect to the classic two player parity games, relies on the fact that it admits a multitude of players. In particular, this is due to the fact that on increasing the number of players, for each state, the number of decisions increases exponentially.

The complexity of the proposed algorithm results to be much better in terms of space consumption. Indeed, it only requires polynomial space, as we prove in the next theorem. In the next section we also show that this is a tight complexity.

Theorem 5.2. *Let $G$ be a* CMPG. *The algorithm $Solve($ $G)$ is computed in space* $|\mathrm{St}|^2 + (log|C| + log|\mathrm{St}| + |\mathrm{Pl}| * log|\mathrm{Ac}|) * |\mathrm{St}|$ *where* St*,* $C$*,* Pl *and* Ac *are the set of states, the set of different priorities, the set of players and the set of the actions, respectively.*

Proof. We first analyze the subroutines of the recursive algorithm. Start with the function $Check(n, f, i, d, s, X)$. At line 2 it is initialized a Boolean variable that needs constant space. For the assignments at the lines 6 and 8, we do not need additional storage space as we can overwrite the same boolean variable. As the FOR construct regards (line 3 - 13) we need $log|\mathrm{Ac}|$ space. Concerning lines 11-13, the algorithm is recursively called (by the next player).

By iterating on the number of players, we have that

$$\begin{aligned}
S_{Check(1)} &= 1 + log|\mathrm{Ac}| \\
S_{Check(2)} &= 1 + log|\mathrm{Ac}| + S_{Check(1)} = 2 * (1 + log|\mathrm{Ac}|) \\
S_{Check(3)} &= 1 + log|\mathrm{Ac}| + S_{Check(2)} = 3 * (1 + log|\mathrm{Ac}|) \\
&\vdots
\end{aligned}$$

The generic element $S_{Check(n-i)}$ is given by

$$S_{Check(n-i)} = \begin{cases} 1 + log|\mathrm{Ac}| & \text{if } i = n-1 \\ 1 + log|\mathrm{Ac}| + S_{Check(n-(i+1))} & \text{if } i < n-1 \end{cases}$$

It is easy to note that, to each iteration, the value $1 + log|\mathrm{Ac}|$ does not depend on the next iteration. So, we can conclude that

$$\begin{aligned}
S_{Check(n)} &= n * (1 + log|\mathrm{Ac}|) \qquad \text{since } n = |\mathrm{Pl}| \\
&= O((1 + log|\mathrm{Ac}|) * |\mathrm{Pl}|) \\
&= O(|\mathrm{Pl}| * log|\mathrm{Ac}|)
\end{aligned}$$

Then, the amount of memory required to perform $Check(n, f, i, d, s, X)$ is $O(|\mathrm{Pl}| * log|\mathrm{Ac}|)$. We proceed by analyzing the space complexity of the function $Attr(G, X, f, n)$. Each of the basic operations needs space $log|\mathrm{St}|$. Then, we have $5 * log|\mathrm{St}|$. By knowing the space complexity of the function $Check(n, f, i, d, s, X)$, we have that the amount of memory required to perform $Attr(G, X, f, n)$ is given by $S_{Attr}(G, X, f, n) = O(log|\mathrm{St}| + |\mathrm{Pl}| * log|\mathrm{Ac}|)$.

Finally, we analyze the complexity of the algorithm $Solve(G)$. The assignments at lines 3-4 require constant space. In addition, the operations from lines 5 to 19 (excluding the attractor calls) need the declaration of the seven variables, each of which needs $|\mathrm{St}|$ space. By knowing the complexity space of the function $Attr(G, X, f, n)$, we have that the space complexity of the recursive algorithm $Solve(G)$ is given by $S_{Solve}(G) = 7*|\mathrm{St}| + log|C| + S_{Solve}(G \setminus A) + S_{Attr}(G, X, f, n)$ where $C$ corresponds to the maximum number of priority.

Let $n = |\mathrm{St}|$, by using an over-approximation of the $G \setminus A$, we have that

$$S_{Solve(n)} = \begin{cases} 2 & \text{if } n = 0 \\ 7*n + log|C| + S_{Solve}(n-1) + S_{Attr}(G, X, f, n) & \text{if } n > 0 \end{cases}$$

By iterating over $n$, we have that

$$\begin{aligned}
S_{Solve}(0) &= 2 \\
S_{Solve}(1) &= 7 + log|C| + S_{Solve}(0) + S_{Attr}(G, X, f, n) \\
&= 7 + log|C| + 2 + S_{Attr}(G, X, f, n) \\
&= 9 + log|C| + S_{Attr}(G, X, f, n) \\
S_{Solve}(2) &= 14 + log|C| + S_{Solve}(1) + S_{Attr}(G, X, f, n) \\
&= 14 + log|C| + 9 + log|C| + S_{Attr}(G, X, f, n) + \\
&\quad S_{Attr}(G, X, f, n) \\
&= 23 + 2 * log|C| + 2 * S_{Attr}(G, X, f, n) \\
&\vdots \\
S_{Solve}(n) &= (7 * (n * (n+1)))/2 + 2 + n * log|C| + n * \\
&\quad S_{Attr}(G, X, f, n) \\
&= (n^2 + (log|C| + S_{Attr}(G, X, f, n)) * n) \\
&= (n^2 + (log|C| + log|\mathrm{St}| + |\mathrm{Pl}| * log|\mathrm{Ac}|) * n) \\
&= (|\mathrm{St}|^2 + (log|C| + log|\mathrm{St}| + |\mathrm{Pl}| * log|\mathrm{Ac}|) * |\mathrm{St}|)
\end{aligned}$$

Hence, in term of space complexity, the algorithm is quadratic in the number of states, logarithmic in the number of priorities and actions and polynomial in the number of players. In other words, it takes polynomial space and therefore we are done with the proof. □

## 6. LOWER BOUND

In this section we show that solving CMPG is PSpace-hard through a reduction from the satisfiability problem of quantified Boolean formulas (QBF, for short). A (fully) quantified Boolean formula is a formula in quantified propositional logic where every variable (from a finite set) is quantified by using either an existential or a universal quantifier. For example, $\exists x \; \forall y \; \exists z \; ((x \wedge z) \vee y)$ is an instance of QBF. A quantified Boolean formula is in *prenex normal form* if it has two basic parts: a portion containing only quantifiers and a portion containing an unquantified Boolean formula. Checking whether in a quantified Boolean formula $\phi$ there exists an assignment profile to the variables, given in accordance with the quantifiers, that satisfies $\phi$ is known to be a PSpace-complete problem. We will provide a reduction such that for each instance $\phi \in$ QBF we build a CMPG $G$ such that $\phi$ is true if and only if the existential coalition in $G$ wins the game.

The reduction we provide uses and extends a reduction introduced in [20, 30]. They show that, in case we restrict to formulas in QBF with a prefix of quantification $\exists\forall$ the satisfiability problem can be reduced to solving a game over a specific concurrent arena (formally a CGS) where each move comes from the possible interaction between the coalitions of players. These moves are defined through a $\Sigma_2^p$ algorithm, named $CPre$, that calculates the pre-image of a set of states over a given coalition. We recall that $\Sigma_i^p$ is the class of problems that can be solved by a nondeterministic Turing machine in polynomial time with $i$ recursive calls to an $NP$ oracle, that $\Delta_{i+1}^p = P^{\Sigma_i^p}$, and that $PH = \bigcup_{k \in \mathrm{N}} \Delta_i^p$ [34].

The $CPre$ algorithm in [20, 30] considers just one call to an $NP$ oracle as the game structure cannot accommodate

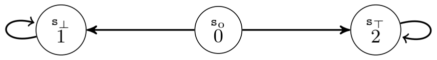

**Figure 6: The game resulting from the reduction.**

more than one coalition alternation. In our case, instead we can address games with an arbitrary number of (strict) alternation among coalitions of players.

Theorem 6.1. *The problem of deciding a winner in a* CMPG *is* PSpace-hard

Proof. Given a fully quantified Boolean formula $\phi = Q_0 x_0 \cdots Q_{n-1} x_{n-1} \varphi(x_0, \ldots, x_{n-1})$ in prenex normal form, we now show how to construct an instance of a CMPG $\mathcal{G}$ such that the existential team wins the game if and only if the formula is satisfiable. The game is $\mathcal{G} \triangleq < \text{Pl}, \text{Ac}, \text{St}, s_I, \lambda,$ $\mathsf{tr} >$ where the set of players $\text{Pl} = \{x_0, \ldots, x_{n-1}\}$ is the set of the Boolean variables, the set of actions $\text{Ac} = \{0, 1\}$ is the sets of the truth values to be assigned to Boolean variables, the set of states is $\text{St} = \{s_0, s_\top, s_\perp\}$, the labeling function $\lambda : \text{St} \to \text{N}$ assigning to each state a priority is such that $\lambda(s_0) = 0$, $\lambda(s_\perp) = 1$, and $\lambda(s_\top) = 2$. Finally, $s_I = \{s_0\} \in \text{St}$ is a designated initial state. It just remains to define the transition function $\mathsf{tr}$. Let $d \in \text{Dc}$ be a specific decision where $\text{Dc} \triangleq \text{Pl} \rightharpoonup \text{Ac}$ is the set of *decisions*. We say that

$$\mathsf{tr}(s_0, d) = \begin{cases} s_\top, & \text{if} \quad d \models \varphi \\ s_\perp, & \textit{otherwise} \end{cases}$$

The game built out of this construction is showed in Figure 6. Note that, if the assignment of the truth values at the Boolean variables makes the formula true then the token remains infinitely often at the state $s_\top$ (so letting the existential team to win the game) otherwise it remains in $s_\perp$ infinitely often.

It is easy to see that by using an explicit transition function (as we have done so far) the reduction from QBF to CMPS is exponential. In order to make the reduction to be polynomial one can use instead an implicit representation of the transition function as in [30]. Precisely, we assume that the transitions from any state $s$ are given by the sequence $((\varphi_0, s_0), (\varphi_1, s_1), \ldots, (\varphi_{|\mathsf{tr}(s)|}, s_{|\mathsf{tr}(s)|}))$, where $s_i \in \text{St}$ and $\varphi_i$ is a boolean combination of propositions $x_j = a_j$ that evaluate to true iff $x_j$ chooses the action $a_j$. The transition table is then defined as follows: $tr(s, a_0, \ldots a_{n-1}) = s_j$ iff $j$ is the lowest index s.t. $\varphi_j$ evaluates to true when players $x_0$ to $x_{n-1}$ choose moves $a_0$ to $a_{n-1}$. At this point, we use the procedure in Figure 7 to calculate the transition function that extends the one introduced in [30]. The procedure $CPre$ guesses an action for each player in order to construct a decision. It expands the set of states that are winning for the existential players if the QBF formula $\varphi$ holds. In particular, note that $s_0$ belongs to $CPre(\text{Pl}, s_\top)$ iff there exists a valuation for variables assigned to the existential quantifiers such that $\varphi_\top$ is true whatever the players associated to the variables universally quantified will choose. Finally, the theorem holds by using the following reasoning. Given $\varphi$, the algorithm $CPre$ requires a tower of guesses whose height depends on the number $n$ of quantifications. More precisely, for all $\varphi$ with $n$ quantifications $CPre$ is in $\Sigma_n^p$.

Hence in general, for each formula $\varphi$, we have that the algorithm $CPre$ is in $PH$.

```
Procedure $CPre(\text{Pl}, \text{S})$:
W = $\emptyset$;
For each $s \in \text{St}$
 $i = 0$;
  For each $i < |\text{Pl}|$
    if ($i$ mod $2 == 0$)
      $a_i = guess_\exists(s, \text{Pl}_i, d_{<i})$;
    else
      $a_i = guess_\forall(s, \text{Pl}_i, d_{<i})$;
    $d_{<i+1} = d_{<i} \cdot a_i$;
    $j = 0$;
    while ($\neg\varphi_j(d)$)
      $j++$;
    if $s_j \in \text{S}$
      $\text{W} = \text{W} \cup \{s_j\}$;
return W
```

**Figure 7: Procedure CPre.**

## 7. CONCLUSION

In this paper, we have introduced and solved *Concurrent Multi-Player Parity Games* (CMPG) as an extension of the classic and well-known (two player turn-based) parity games. In a CMPG several players concurrently and independently move along edges by taking decisions as tuples of action moves. As for classic parity games the arena is labeled with natural number priorities. In a CMPG players are partitioned into two teams: the existential and universal teams. In a play, players take role (from the two teams) in a strict $\exists\forall$ alternation and the existential team wins the game if all the players it comprises can enforce a play that satisfies the parity condition, i.e. the maximal priority that is seen infinitely often is even. We have proved that the problem of solving a CMPG is PSpace-complete. For the upper bound we have introduced a non-trivial extension of the *attractor* function along the Zielonka Recursive Algorithm. For the lower bound we have provided a reduction from the QBF satisfiability problem.

Classic parity games have been deeply investigated in a number of application fields both as an abstract tool to model reactive systems and a machinery to properly solve decision problems. In the multi-player setting they have been also recently applied to reasoning about the strategic interaction of players by means of logic formalisms [5]. However, a strong limitation in this area resides on the fact that problems have to be first casted somehow in a two-player turn-based framework and then reduced to parity games. This often requires resources and the solution through these games becomes less effective. We believe that having now introduced and solved efficiently concurrent multi-player parity games could get back more attention to such an approach and solve open problems efficiently in the multi-player setting.

As future work it would be interesting to consider some improvements and heuristic evaluation by implementing our algorithm. A good platform to do this is PGSolver [17] that contains most of the known algorithms to solve classic parity games.

## Acknowledgments

This work has been partially supported by the FP7 EU project 600958-SHERPA and the IndAM project "Logica, Automi e Giochi per Sistemi Auto-adattivi".

## REFERENCES

[1] R. Alur, T. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *JACM*, 49(5):672–713, 2002.

[2] B. Aminof, F. Mogavero, and A. Murano. Synthesis of Hierarchical Systems. *SCP*, 83:56–79, 2014.

[3] B. Aminof and O. Kupferman and A. Murano. Improved model checking of hierarchical systems. *Inf. Comput.*, 210:68–86, 2012.

[4] D. Berwanger. Admissibility in Infinite Games. In *STACS'07*, LNCS 4393, pages 188–199. Springer, 2007.

[5] P. Cermak, A. Lomuscio, and A. Murano. Verifying and synthesising multi-agent systems against one-goal strategy logic specifications. In *AAAI'15*, pages 2038–2044, 2015.

[6] K. Chatterjee, L. D. Alfaro, and T. Henzinger. The complexity of quantitative concurrent parity games. In *SODA'06*, pages 678–687, 2006.

[7] K. Chatterjee, L. D. Alfaro, and T. A. Henzinger. Qualitative concurrent parity games. *TOCL*, 12(4):28, 2011.

[8] K. Chatterjee, L. Doyen, T. Henzinger, and J.-F. Raskin. Generalized Mean-payoff and Energy Games. In *FSTTCS'10*, LIPIcs 8, pages 505–516. Leibniz-Zentrum fuer Informatik, 2010.

[9] K. Chatterjee, T. Henzinger, and M. Jurdzinski. Mean-Payoff Parity Games. In *LICS'05*, pages 178–187. IEEE Computer Society, 2005.

[10] K. Chatterjee, T. Henzinger, and N. Piterman. Strategy Logic. *Information and Computation*, 208(6):677–693, 2010.

[11] K. Chatterjee and T. A. Henzinger. A survey of stochastic $\omega$-regular games. *Journal of Computer and System Sciences*, 78(2):394–413, 2012.

[12] K. Chatterjee, M. Jurdziński, and T. Henzinger. Quantitative stochastic parity games. In *SODA'04*, pages 121–130, 2004.

[13] E. Clarke and E. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *LP'81*, LNCS 131, pages 52–71. Springer, 1981.

[14] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 2002.

[15] E. Emerson and C. Jutla. The Complexity of Tree Automata and Logics of Programs (Extended Abstract). In *FOCS'88*, pages 328–337. IEEE Computer Society, 1988.

[16] O. Friedmann. Recursive algorithm for parity games requires exponential time. *RAIRO-Theoretical Informatics and Applications*, 45(04):449–457, 2011.

[17] O. Friedmann and M. Lange. Solving parity games in practice. In *ATVA'09*, LNCS 5799, pages 182–196. Springer, 2009.

[18] D. Harel and A. Pnueli. *On the Development of Reactive Systems.* Springer, 1985.

[19] P. Hoffmann and M. Luttenberger. Solving parity games on the GPU. In *ATVA'13*, pages 455–459. Springer, 2013.

[20] W. Jamroga and J. Dix. Do agents make model checking explode (computationally)? In *CEEMAS'15*, LNCS 3690, pages 398–407. Springer, 2005.

[21] W. Jamroga and A. Murano. On Module Checking and Strategies. In *AAMAS'14*, pages 701–708. IFAAMAS, 2014.

[22] W. Jamroga and A. Murano. Module checking of strategic ability. In *AAMAS'15*, pages 227–235. IFAAMAS, 2015.

[23] M. Jurdzinski. Deciding the winner in parity games is in up $\cap$ co-up. *Inf. Process. Lett.*, 68(3):119–124, 1998.

[24] M. Jurdzinski. Small progress measures for solving parity games. In *STACS'00*, LNCS 1770, pages 290–301. Springer, 2000.

[25] M. Jurdzinski, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. Comput.*, 38(4):1519–1532, 2008.

[26] O. Kupferman, M. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *JACM*, 47(2):312–360, 2000.

[27] O. Kupferman, M. Vardi, and P. Wolper. Module Checking. *Information and Computation*, 164(2):322–344, 2001.

[28] O. Kupferman and M. Y. Vardi. Weak alternating automata and tree automata emptiness. In *STOC'98*, pages 224–233. ACM, 1998.

[29] F. Laroussinie and N. Markey. Augmenting atl with strategy contexts. *Information and Computation*, 245:98–123, 2015.

[30] F. Laroussinie, N. Markey, and G. Oreiby. On the expressiveness and complexity of atl. *LMCS*, 4(2):1–25, 2008.

[31] A. Lopes, F. Laroussinie, and N. Markey. ATL with Strategy Contexts: Expressiveness and Model Checking. In *FSTTCS'10*, LIPIcs 8, pages 120–132. Leibniz-Zentrum fuer Informatik, 2010.

[32] F. Mogavero, A. Murano, G. Perelli, and M. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *TOCL*, 15(4):34:1–42, 2014.

[33] F. Mogavero, A. Murano, and M. Vardi. Reasoning About Strategies. In *FSTTCS'10*, LIPIcs 8, pages 133–144. Leibniz-Zentrum fuer Informatik, 2010.

[34] C. H. Papadimitriou. *Computational complexity.* John Wiley and Sons Ltd., 2003.

[35] J. Queille and J. Sifakis. Specification and Verification of Concurrent Programs in Cesar. In *SP'81*, LNCS 137, pages 337–351. Springer, 1981.

[36] S. Schewe. Solving parity games in big steps. In *FSTTCS'07*, LNCS 4855, pages 449–460. Springer, 2007.

[37] A. D. Stasio, A. Murano, V. Prignano, and L. Sorrentino. Solving parity games in scala. In *FACS'14*, LNCS 8997, pages 145–161. Springer, 2015.

[38] W. Thomas. Infinite games and verification. In *CAV'02*, pages 58–64. Springer, 2002.

[39] M. Yannakakis and K. Etessami. Recursive concurrent stochastic games. *Logical Methods in Computer Science*, 4, 2008.

[40] W. Zielonka. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *TCS*, 200(1-2):135–183, 1998.