

Partial solvers for generalized parity games

Véronique Bruyère (UMONS)
Jean-François Raskin (ULB)

Guillermo A. Pérez (UAntwerp)
Clément Tamines (UMONS)

September 11, 2019
Reachability Problems '19
Brussels, Belgium

Outline of the talk

Recall the **significance of parity games**

- LTL synthesis : classical method to perform LTL synthesis using parity games
- difficulties of this method

Introduce **generalized parity games**

- link with LTL synthesis
- the difficulties of solving such games

Discuss methods used to **eliviate these difficulties**

- **partial solvers** and their **adaptation** to generalized parity games
- our **experimental results**

Reactive synthesis from LTL specifications

Parity games are used as **intermediary steps** to solve the reactive synthesis problem from LTL specifications [7]

Input : an LTL formula ϕ whose propositional variables are partitioned into inputs (controllable by the environment) and outputs (controlled by the system)

Classical algorithm :

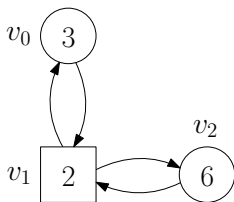
- construct a deterministic parity automaton (DPA)
- the DPA can then be seen as a two player graph game
- winning condition in this game : parity acceptance condition of the DPA

Difficulties : DPA may be doubly exponential in the size of ϕ and its parity condition may require exponentially many priorities

→ underlying parity game may be hard to solve !

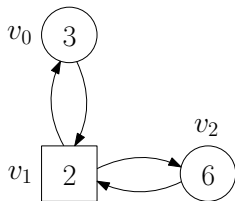
Game structure and play

Game structure $G = (V_0, V_1, E)$ and priority function $\alpha: V \rightarrow \{0, 1, \dots, d\}$



Game structure and play

Game structure $G = (V_0, V_1, E)$ and priority function $\alpha: V \rightarrow \{0, 1, \dots, d\}$

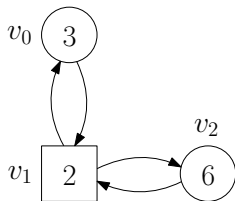


$$\pi = v_0 v_1 v_2 (v_1 v_0)^\omega$$

Play: infinite path $\pi = v_0 v_1 \dots \in V^\omega$ in G starting in some initial vertex v_0

Game structure and play

Game structure $G = (V_0, V_1, E)$ and priority function $\alpha: V \rightarrow \{0, 1, \dots, d\}$



$$\pi = v_0 v_1 v_2 (v_1 v_0)^\omega$$

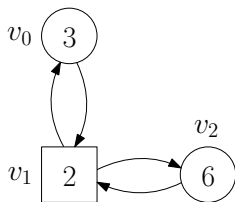
$$\alpha(\pi) = 3 \ 2 \ 6 \ (2 \ 3)^\omega$$

Play: infinite path $\pi = v_0 v_1 \dots \in V^\omega$ in G starting in some initial vertex v_0

→ we consider the **sequence of priorities** corresponding to π

Game structure and play

Game structure $G = (V_0, V_1, E)$ and priority function $\alpha: V \rightarrow \{0, 1, \dots, d\}$



$$\pi = v_0 v_1 v_2 (v_1 v_0)^\omega$$

$$\alpha(\pi) = 3 \ 2 \ 6 \ (2 \ 3)^\omega$$

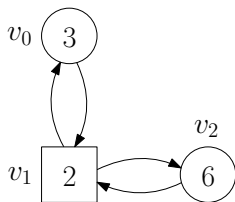
Play: infinite path $\pi = v_0 v_1 \dots \in V^\omega$ in G starting in some initial vertex v_0

→ we consider the **sequence of priorities** corresponding to π

Objective for player $i \in \{0, 1\}$: set of plays $\Omega_i \subseteq V^\omega$

Game structure and play

Game structure $G = (V_0, V_1, E)$ and priority function $\alpha: V \rightarrow \{0, 1, \dots, d\}$



$$\pi = v_0 v_1 v_2 (v_1 v_0)^\omega$$

$$\alpha(\pi) = 3 \ 2 \ 6 \ (2 \ 3)^\omega$$

Play: infinite path $\pi = v_0 v_1 \dots \in V^\omega$ in G starting in some initial vertex v_0

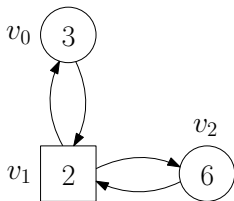
→ we consider the **sequence of priorities** corresponding to π

Objective for player $i \in \{0, 1\}$: set of plays $\Omega_i \subseteq V^\omega$

- $\Omega_0 = \text{0Parity}(\alpha)$: plays where maximum priority seen infinitely often is **even**

Game structure and play

Game structure $G = (V_0, V_1, E)$ and priority function $\alpha: V \rightarrow \{0, 1, \dots, d\}$



$$\pi = v_0 v_1 v_2 (v_1 v_0)^\omega$$

$$\alpha(\pi) = 3 \ 2 \ 6 \ (2 \ 3)^\omega$$

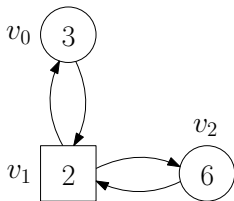
Play: infinite path $\pi = v_0 v_1 \dots \in V^\omega$ in G starting in some initial vertex v_0
 \rightarrow we consider the **sequence of priorities** corresponding to π

Objective for player $i \in \{0, 1\}$: set of plays $\Omega_i \subseteq V^\omega$

- $\Omega_0 = 0\text{Parity}(\alpha)$: plays where maximum priority seen infinitely often is **even**
- $\Omega_1 = 1\text{Parity}(\alpha)$: plays where maximum priority seen infinitely often is **odd**

Strategy

A play π is **winning** for player 0 (resp. 1) if $\pi \in \Omega_0$ (resp. Ω_1), and losing otherwise

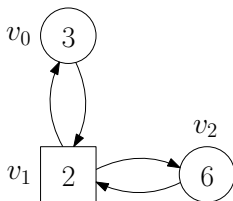


$$\pi = v_0 v_1 v_2 (v_1 v_0)^\omega$$

$$\alpha(\pi) = 3 2 6 (2 3)^\omega$$

Strategy

A play π is **winning** for player 0 (resp. 1) if $\pi \in \Omega_0$ (resp. Ω_1), and losing otherwise



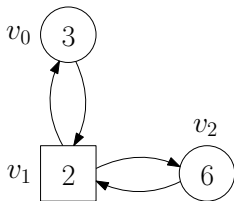
$$\pi = v_0 v_1 v_2 (v_1 v_0)^\omega$$

$$\alpha(\pi) = 3 \ 2 \ 6 \ (2 \ 3)^\omega$$

→ play π is winning for player 1

Strategy

A play π is **winning** for player 0 (resp. 1) if $\pi \in \Omega_0$ (resp. Ω_1), and losing otherwise



$$\pi = v_0 v_1 v_2 (v_1 v_0)^\omega$$

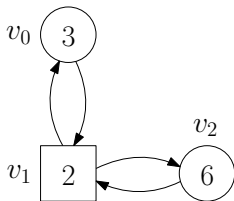
$$\alpha(\pi) = 3 \ 2 \ 6 \ (2 \ 3)^\omega$$

→ play π is winning for player 1

Solving G: finding $\text{Win}(G, i, \Omega_i)_{i \in \{0,1\}} \subseteq V$ from which player i can ensure Ω_i

Strategy

A play π is **winning** for player 0 (resp. 1) if $\pi \in \Omega_0$ (resp. Ω_1), and losing otherwise



$$\pi = v_0 v_1 v_2 (v_1 v_0)^\omega$$

$$\alpha(\pi) = 3 \ 2 \ 6 \ (2 \ 3)^\omega$$

→ play π is winning for player 1

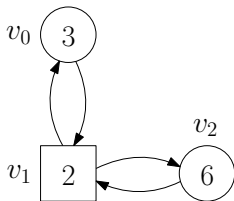
Solving G: finding $\text{Win}(G, i, \Omega_i)_{i \in \{0,1\}} \subseteq V$ from which player i can ensure Ω_i

Strategy: $V^* \times V_0 \rightarrow V$ which dictates player i 's choices

Given $\pi = \underbrace{v_0 v_1 \dots v_t}_h \underbrace{v_{t+1}}_{\in V_i}$ decides v_{t+1} from $h v_t$ (finite-memory) or v_t (without)

Strategy

A play π is **winning** for player 0 (resp. 1) if $\pi \in \Omega_0$ (resp. Ω_1), and losing otherwise



$$\pi = v_0 v_1 v_2 (v_1 v_0)^\omega$$

$$\alpha(\pi) = 3 \ 2 \ 6 \ (2 \ 3)^\omega$$

→ play π is winning for player 1

Solving G: finding $\text{Win}(G, i, \Omega_i)_{i \in \{0,1\}} \subseteq V$ from which player i can ensure Ω_i

Strategy: $V^* \times V_0 \rightarrow V$ which dictates player i 's choices

Given $\pi = \underbrace{v_0 v_1 \dots v_t}_h \underbrace{v_{t+1}}_{\in V_i}$ decides v_{t+1} from $h v_t$ (finite-memory) or v_t (without)

Complexity: $\text{NP} \cap \text{coNP}$ [2] and $\text{UP} \cap \text{coUP}$ [6], P membership is open problem

Back to reactive synthesis

If the LTL formula ϕ is a **conjunction of smaller formulas** : $\phi = \phi_1 \wedge \phi_2 \cdots \wedge \phi_n$

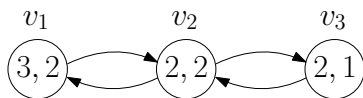
- we can perform a compositional approach
- the procedure constructs a DPA A_i **for each** subformula ϕ_i
- underlying game is then the product of the automata A_i and the winning condition is a **conjunction (for player 0) of parity conditions**

→ those games are **generalized parity games**

Generalized parity objective

Now consider $k \geq 1$ priority functions $\alpha_\ell: V \rightarrow \{0, \dots, d_\ell\} \forall \ell \in \{1, \dots, k\}$
 \rightarrow we consider several sequences of priorities corresponding to a play

Objective $\Omega_0 = \text{ConjEvenParity}(\alpha_1, \dots, \alpha_k)$, conjunction of parity objectives :
 maximum priority seen infinitely often according to every function is even
 Opposite $\Omega_1 = \text{DisjOddParity}(\alpha_1, \dots, \alpha_k)$, disjunction of parity objective :
 maximum priority seen infinitely often according to some function is odd



$$\pi = v_1 (v_2 v_3)^\omega$$

$$\alpha_1(\pi) = 3 (2 2)^\omega$$

$$\alpha_2(\pi) = 2 (2 1)^\omega$$

Specificity of generalized parity games :

- coNP – complete [1]
- player 0 requires a finite-memory strategy to win

Partial solvers

Incomplete algorithms that **partially solve** (generalized) parity games

- for parity objective : existing **polynomial-time** algorithms [3, 4, 9]
- experimentally shown to **behave well** on random benchmarks and completely solve structured benchmarks from PGSolver [3]

Formally, a partial solver returns

- **subsets of winning nodes** $Z_0 \subseteq \text{Win}(G, 0, \Omega_0)$ and $Z_1 \subseteq \text{Win}(G, 1, \Omega_1)$
- **a sub-game** $G \setminus (Z_0 \cup Z_1)$ that is **unsolved**

Our contributions

- **extend** the partial solvers for parity games to generalized parity games
- **combine** partial solvers with recursive complete algorithms [10, 1]
- **evaluate** on benchmarks generated from LTL formulas

Partial solver using Büchi games

A first very simple partial solver based on Büchi and safety objectives

Based on a simple remark for player 0 (symmetric for player 1)

- if player 0 can ensure to visit infinitely often an even priority
- without visiting a greater odd priority
- then he is winning for $\text{OParity}(\alpha)$

Partial solver using Büchi games

A first very simple partial solver based on Büchi and safety objectives

Based on a simple remark for player 0 (symmetric for player 1)

- if player 0 can ensure to visit infinitely often an even priority
- without visiting a greater odd priority
- then he is winning for $0\text{Parity}(\alpha)$

Given $p \in \{0, \dots, d\}$ an even priority, we want to

Partial solver using Büchi games

A first very simple partial solver based on Büchi and safety objectives

Based on a simple remark for player 0 (symmetric for player 1)

- if player 0 can ensure to visit infinitely often an even priority
- without visiting a greater odd priority
- then he is winning for $0\text{Parity}(\alpha)$

Given $p \in \{0, \dots, d\}$ an even priority, we want to

- visit $U = \{v \in V \mid \alpha(v) = p\}$ infinitely often: **Buchi(U)**

Partial solver using Büchi games

A first very simple partial solver based on Büchi and safety objectives

Based on a simple remark for player 0 (symmetric for player 1)

- if player 0 can ensure to visit infinitely often an even priority
- without visiting a greater odd priority
- then he is winning for $0\text{Parity}(\alpha)$

Given $p \in \{0, \dots, d\}$ an even priority, we want to

- visit $U = \{v \in V \mid \alpha(v) = p\}$ infinitely often: **Buchi(U)**
- avoid $U' = \{v \in V \mid \alpha(v) \text{ is an odd priority and } \alpha(v) > p\}$: **Safe(U')**

Partial solver using Büchi games

A first very simple partial solver based on Büchi and safety objectives

Based on a simple remark for player 0 (symmetric for player 1)

- if player 0 can ensure to visit infinitely often an even priority
- without visiting a greater odd priority
- then he is winning for $0\text{Parity}(\alpha)$

Given $p \in \{0, \dots, d\}$ an even priority, we want to

- visit $U = \{v \in V \mid \alpha(v) = p\}$ infinitely often: $\text{Buchi}(U)$
- avoid $U' = \{v \in V \mid \alpha(v) \text{ is an odd priority and } \alpha(v) > p\}$: $\text{Safe}(U')$

Conjunction of Büchi and safety objectives

If $v \in \text{Win}(G, 0, \text{Buchi}(U) \cap \text{Safe}(U'))$, then $v \in \text{Win}(G, 0, 0\text{Parity}(\alpha))$.

Partial solver using Büchi games

A first very simple partial solver based on Büchi and safety objectives

Based on a simple remark for player 0 (symmetric for player 1)

- if player 0 can ensure to visit infinitely often an even priority
- without visiting a greater odd priority
- then he is winning for $0\text{Parity}(\alpha)$

Given $p \in \{0, \dots, d\}$ an even priority, we want to

- visit $U = \{v \in V \mid \alpha(v) = p\}$ infinitely often: $\text{Buchi}(U)$
- avoid $U' = \{v \in V \mid \alpha(v) \text{ is an odd priority and } \alpha(v) > p\}$: $\text{Safe}(U')$

Conjunction of Büchi and safety objectives

If $v \in \text{Win}(G, 0, \text{Buchi}(U) \cap \text{Safe}(U'))$, then $v \in \text{Win}(G, 0, 0\text{Parity}(\alpha))$.

Variant : property holds with $\text{CoBuchi}(U')$ instead of $\text{Safe}(U')$

Partial solver using Büchi games

Algorithm 1 BuchiSolver(G, α)

for each $p \in \{0, \dots, d\}$ **do**

$i = p \bmod 2$

$U = \{v \in V \mid \alpha(v) = p\}$

$U' = \{v \in V \mid \alpha(v) \text{ is a } (1 - i)\text{-priority and } \alpha(v) > p\}$

$W = \text{Win}(G, i, \text{Buchi}(U) \cap \text{Safe}(U'))$

if $W \neq \emptyset$ **then**

$\{Z_i, Z_{1-i}\} = \text{BuchiSolver}(G \setminus W, \alpha)$

return $\{Z_i \cup W, Z_{1-i}\}$

return $\{\emptyset, \emptyset\}$

- considers **all priorities** and **both players**
- is **recursively called on sub-game** when some winning nodes are found

Partial solver using generalized Büchi games

Adaptation to generalized parity games based on **generalized Büchi** objective

Player 0 with the **conjunction** of parity objectives, **on all dimensions l** he wants

- to ensure to visit infinitely often an even priority p_l
- without visiting an odd priority greater than p_l
- if he does, he is winning for $\text{ConjEvenParity}(\alpha_1, \dots, \alpha_k)$

Partial solver using generalized Büchi games

Adaptation to generalized parity games based on **generalized Büchi** objective

Player 0 with the **conjunction** of parity objectives, **on all dimensions l** he wants

- to ensure to visit infinitely often an even priority p_l
- without visiting an odd priority greater than p_l
- if he does, he is winning for $\text{ConjEvenParity}(\alpha_1, \dots, \alpha_k)$

→ this is **Generalized Büchi objective** using k sets $U_l = \{v \in V \mid \alpha_l(v) = p_l\}$

Partial solver using generalized Büchi games

Adaptation to generalized parity games based on **generalized Büchi** objective

Player 0 with the **conjunction** of parity objectives, **on all dimensions l** he wants

- to ensure to visit infinitely often an even priority p_l
- without visiting an odd priority greater than p_l
- if he does, he is winning for $\text{ConjEvenParity}(\alpha_1, \dots, \alpha_k)$

→ this is **Generalized Büchi objective** using k sets $U_l = \{v \in V \mid \alpha_l(v) = p_l\}$

Player 1 with the **disjunction** of parity objectives, **on some dimensions l**

- if he ensures $\text{1Parity}(\alpha_l)$, he satisfies the disjunction

Partial solver using generalized Büchi games

Adaptation to generalized parity games based on **generalized Büchi** objective

Player 0 with the **conjunction** of parity objectives, **on all dimensions l** he wants

- to ensure to visit infinitely often an even priority p_l
- without visiting an odd priority greater than p_l
- if he does, he is winning for $\text{ConjEvenParity}(\alpha_1, \dots, \alpha_k)$

→ this is **Generalized Büchi objective** using k sets $U_l = \{v \in V \mid \alpha_l(v) = p_l\}$

Player 1 with the **disjunction** of parity objectives, **on some dimensions l**

- if he ensures $\text{1Parity}(\alpha_l)$, he satisfies the disjunction

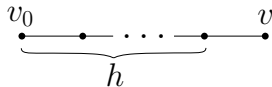
→ can apply **same property** as before **dimension by dimension**

Partial solver : Good Episode

GoodEpSolver [4] for parity games

For player 0 in a parity game (symmetric for player 1)

- if he can ensure to visit some vertex v from v_0
- we consider the consistent history hv from v_0 to v



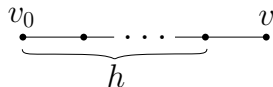
If the maximum visited priority in h is even, h is a **good episode**

Partial solver : Good Episode

GoodEpSolver [4] for parity games

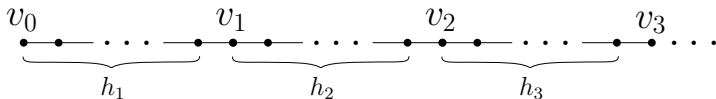
For player 0 in a parity game (symmetric for player 1)

- if he can ensure to visit some vertex v from v_0
- we consider the consistent history hv from v_0 to v



If the maximum visited priority in h is even, h is a **good episode**

If a play consists in a **succession of good episodes**, it is won by player 0

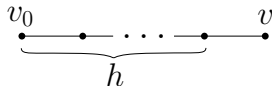


Partial solver : Good Episode

GoodEpSolver [4] for parity games

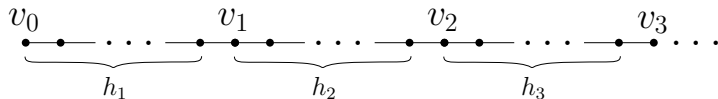
For player 0 in a parity game (symmetric for player 1)

- if he can ensure to visit some vertex v from v_0
- we consider the consistent history hv from v_0 to v



If the maximum visited priority in h is even, h is a **good episode**

If a play consists in a **succession of good episodes**, it is won by player 0

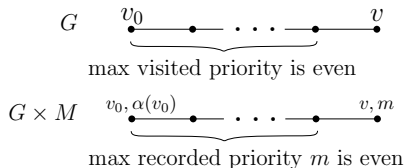


Maximum priority in each h_j is even \rightarrow objective $0\text{Parity}(\alpha)$ is satisfied

Partial solver : Good episode

To compute good episodes: introduce an **extended game structure** $G \times M$

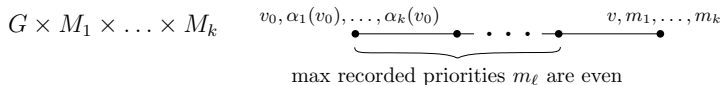
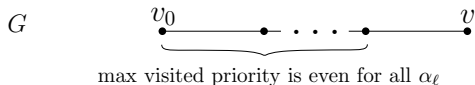
→ records the maximal visited priority **by construction**



Generalized good episode

The generalized version computes a specific fixpoint **depending on the player**

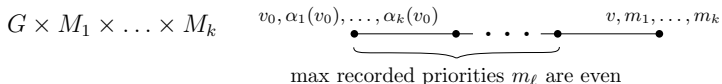
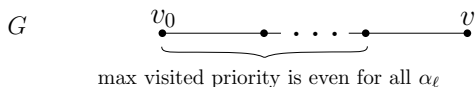
Player 0: **extended game structure** $G \times M_1 \times \dots \times M_k$



Generalized good episode

The generalized version computes a specific fixpoint **depending on the player**

Player 0: **extended game structure** $G \times M_1 \times \dots \times M_k$

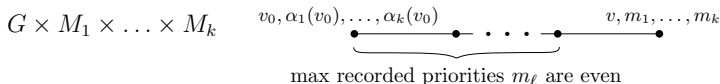
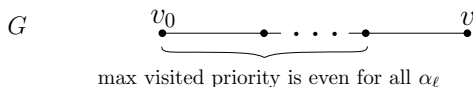


Good episode: history h s.t. **for all α_ℓ** the maximum priority visited along h is even
 → ensure a succession of good episodes by computing similar fixpoint

Generalized good episode

The generalized version computes a specific fixpoint **depending on the player**

Player 0: **extended game structure** $G \times M_1 \times \dots \times M_k$



Good episode: history h s.t. **for all α_ℓ** the maximum priority visited along h is even

→ ensure a succession of good episodes by computing similar fixpoint

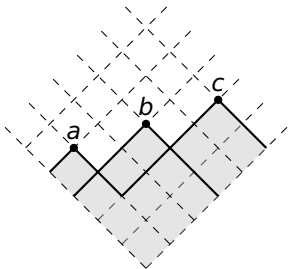
Player 1: **similar to previous approach**

→ compute $\text{GoodEp}_1(G, \alpha_\ell)$ **for some dimension ℓ**

Antichains

Explicit construction and attractor computation in the extended game is expensive
→ introduction of antichains to reduce required space

- let S be a set of elements with \preceq a partial order
- S is closed if whenever $s \in S$ and $s' \preceq s$, then $s' \in S$
- canonical representation of S by the antichain $[S]$ of its maximal elements
- example : grey zone S represented by the antichain $\{a, b, c\}$



Antichains in the extended game

Introduce a **partial order** \preceq on the nodes (v, m) of $G \times M$

Example : if $d = 4$, then $(v, 4) \preceq (v, 2) \preceq (v, 0) \prec (v, 1) \preceq (v, 3)$

Property : if player 0 wins from (v, m) then he wins from $(v', m') \preceq (v, m)$

F_0 is **closed** and the operations used to get F_{j+1} from F_j **preserve closedness**

- every set used to compute fixpoint $\text{GoodEp}_0(G, \alpha)$ are closed
- can therefore be represented by the **antichain of their maximal elements**

Improved version of $\text{GoodEp}_0(G, \alpha)$ computation that **only** handles antichains

- sets manipulated are smaller as these antichains contain **less elements**
- using solely antichain operations we **avoid constructing** $G \times M$

Combining with complete solvers

We adapted a third partial solver [3] to generalized parity games

Recursive algorithm [10] (complete solver) performs well in practice for parity [8]

Partial solvers can be **combined with the recursive algorithm**, given parity game G

- first execute the partial solver on G
- if the game is completely solved, return solutions
- else execute a step of recursive algorithm on the unsolved subgame G'
- return union of the partial solutions with the solutions obtained for G'

Holds for the adaptation of the recursive algorithm to generalized parity games [1]

Combining with complete solvers

We adapted a third partial solver [3] to generalized parity games

Recursive algorithm [10] (complete solver) performs well in practice for parity [8]

Partial solvers can be **combined with the recursive algorithm**, given parity game G

- first execute the partial solver on G
- if the game is completely solved, return solutions
- else execute a step of recursive algorithm on the unsolved subgame G'
- return union of the partial solutions with the solutions obtained for G'

Holds for the adaptation of the recursive algorithm to generalized parity games [1]

Our experimental results

- implemented **6 algorithms** and their **combination with recursive algorithms**
- evaluated on benchmarks **generated from meaningful LTL specifications** [5]

Results for parity games

240 benchmarks with mean size $|V|$ of around 46K and maximal size of 3157K, and a mean number d of priorities of 4.1 with a maximal number $d = 15$

Solver	Solved	T.O.	Fastest	Mean time (233)
Zielonka	240 (100%)	0	150 (62 %)	272 ms
Ziel&BuchiSolver	240 (100%)	0	89 (37 %)	480 ms
Ziel&GoodEpSolver	233 (97%)	7	0 (0%)	1272 ms
Ziel&LaySolver	238 (99%)	2	1 (1%)	587 ms
BuchiSolver	203 (84%) - 37	0	-	-
GoodEpSolver	233 (97%) - 0	7	-	-
LaySolver	232 (97%) - 6	2	-	-

- Algorithm Zielonka is faster than partial solvers on average which was not observed on random graphs in [4]
- Our experiments also show that their combination with a partial solver improves the performances of Algorithms Zielonka 90 cases over 240 (38%)

Results for generalized parity games

152 benchmarks with mean size $|V|$ of around 207K and maximal size of 709K, mean number of priority functions is 4.53 and maximum number is 17.

Solver	Solved	T.O.	Fastest	Mean-Time (87)
GenZielonka	128 (84%)	24	33 (25%)	66 ms
GenZiel&GenBuchiSolver	130 (86%)	22	72 (55%)	56 ms
GenZiel&GenGoodEpSolver	112 (74%)	40	24 (18%)	644 ms
GenZiel&GenLaySolver	110 (72%)	42	3 (2%)	1133 ms
GenBuchiSolver	110 (72%) - 20	22	-	-
GenGoodEpSolver	112 (74%) - 0	40	-	-
GenLaySolver	104 (68%) - 6	42	-	-

- some benchmarks can't be solved by GenZielonka or a partial solver alone, but can be solved by the combination of GenZielonka with a partial solver
- combination with a partial solver improves the performances of GenZielonka in 99 cases over 132 (75%)

- [1] K. Chatterjee, T. A. Henzinger, and N. Piterman.
Generalized parity games.
In *FOSSACS Proceedings*, volume 4423 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2007.
- [2] E. A. Emerson and C. S. Jutla.
Tree automata, mu-calculus and determinacy (extended abstract).
In *FOCS Proceedings*, pages 368–377. IEEE Computer Society, 1991.
- [3] M. Huth, J. H. Kuo, and N. Piterman.
Fatal attractors in parity games: Building blocks for partial solvers.
CoRR, abs/1405.0386, 2014.
- [4] M. Huth, J. H. Kuo, and N. Piterman.
Static analysis of parity games: Alternating reachability under parity.
In *Semantics, Logics, and Calculi - Essays Dedicated to Hanne Riis Nielson and Flemming Nielson on the Occasion of Their 60th Birthdays*, volume 9560 of *Lecture Notes in Computer Science*, pages 159–177. Springer, 2016.

- [5] S. Jacobs, R. Bloem, M. Colange, P. Faymonville, B. Finkbeiner, A. Khalimov, F. Klein, M. Luttenberger, P. J. Meyer, T. Michaud, M. Sakr, S. Sickert, L. Tentrup, and A. Walker.

The 5th reactive synthesis competition (SYNTCOMP 2018): Benchmarks, participants & results.

CoRR, abs/1904.07736, 2019.

- [6] M. Jurdzinski.

Deciding the winner in parity games is in $UP \cap co-UP$.

Inf. Process. Lett., 68(3):119–124, 1998.

- [7] A. Pnueli and R. Rosner.

On the synthesis of a reactive module.

In *POPL Proceedings*, pages 179–190. ACM Press, 1989.

[8] T. van Dijk.

Oink: An implementation and evaluation of modern parity game solvers.

In *TACAS Proceedings, Part I*, volume 10805 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2018.

[9] S. Vester.

Winning cores in parity games.

In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 662–671, 2016.

[10] W. Zielonka.

Infinite games on finitely coloured graphs with applications to automata on infinite trees.

Theor. Comput. Sci., 200(1-2):135–183, 1998.