Explicit and Symbolic Approaches for Parity Games

Antonio Di Stasio

Department of Computer Science, University of Oxford, UK

Abstract

In this paper, we review a broad investigation of the symbolic approach for solving Parity Games. Specifically, we implement in a tool, called `SymPGSolver`, four symbolic algorithms to solve Parity Games and compare their performances to the corresponding explicit versions for different classes of games. By means of benchmarks, we show that for random games, even for constrained random games, explicit algorithms actually perform better than symbolic algorithms. The situation changes, however, for structured games, where symbolic algorithms seem to have the advantage. This suggests that when evaluating algorithms for parity-game solving, it would be useful to have real benchmarks and not only random benchmarks, as the common practice has been.

Keywords

Parity Games, Symbolic Algorithms

Parity games (PGs) [1] are abstract games with a key role in automata theory and formal verification [2, 3, 4, 5, 6]. In the basic setting, parity games are two-player, turn-based, played on directed graphs whose nodes are labeled with priorities (also called, *colors*) and players have perfect information about the adversary moves. The two players, Player 0 and Player 1, take turns moving a token along the edges of the graph starting from a designated initial node. Thus, a play induces an infinite path and Player 0 wins the play if the smallest priority visited infinitely often is even; otherwise, Player 1 wins the play.

In formal system design [7, 3, 5, 8] parity games arise as a natural evaluation machinery for the automatic synthesis and verification of distributed and reactive systems [9, 10, 11], as they allow to express liveness and safety properties in a very elegant and powerful way [12]. Specifically, in model-checking, one can check the correctness of a system with respect to a desired behavior, that is, a *Kripke structure*, by checking whether a model of the system is correct with respect to a formal specification of its behavior. In case the specification is given as a μ-calculus formula [13], the model checking question can be rephrased, in linear-time, as a parity game [1]. Then, a parity game solver can be used as a model checker for a μ-calculus specification (and vice-versa), as well as for fragments such as CTL, CTL*, and the like.

In the automata-theoretic approach to μ-calculus model checking, under a linear-time translation, one can also reduce the verification problem to a question about automata. More precisely, one can take the product of the model and an alternating tree automaton accepting all tree models of the specification. This product can be defined as an alternating word parity automaton

IPS-RiCeRcA-SPIRIT 2022: 10th Italian Workshop on Planning and Scheduling, RiCeRcA Italian Workshop, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy.

antonio.distasio@cs.ox.uk (A. Di Stasio)
https://antoniodistasio.github.io/ (A. Di Stasio)
0000-0001-5475-2978 (A. Di Stasio)

over a singleton alphabet, and the system is correct with respect to the specification iff this automaton is nonempty [5]. It has been proved there that the nonemptiness problems for nondeterministic tree parity automata and alternating word parity automata over a singleton alphabet are equivalent and that their complexities coincide. Hence, algorithms for the solution of the μ-calculus model checking problem, parity games, and the emptiness problem for parity automata can be interchangeably used to solve any of these problems, as they are linear-time equivalent.

The problem of deciding if Player 0 has a winning strategy (i.e., can induce a winning play) in a given parity game is known to be in UPTIME $\cap$ CoUPTIME [14]; whether a polynomial time solution exists is a long-standing open question [6]. Several algorithms to solve PGs have been proposed aiming to tighten the asymptotic complexity of the problem, as well as to work well in practice. Well known are *Recursive* (`RE`) [15], small-progress measures (`SPM`) [16], and `APT` [2, 17], the latter originated to deal with the emptiness of parity automata. Recently, Calude et al. [18] have given a major breakthrough providing a quasi-polynomial time algorithm for solving parity games that runs in time $O(n^{\lceil log(c)+6 \rceil})$. Previously, the best known algorithm for parity games was Dominion Decomposition [19] which could solve parity games in $O(n^{\sqrt{n}})$, so this new result represents a significant advance in the understanding of parity games. Notably, all these algorithms are *explicit*, that is, they are formulated in terms of the underlying game graphs. Due to the exponential growth of finite-state systems, and, consequently, of the corresponding game graphs, the state-explosion problem limits the scalability of these algorithms in practice. Hence for the analysis of large finite-state systems symbolic algorithms are necessary.

Symbolic algorithms are an efficient way to deal with extremely large graphs. They avoid explicit access to graphs by using a set of predefined operations that manipulate Binary Decision Diagrams (BDDs) [20] representing these graphs. This enables handling large graphs succinctly, and, in general, it makes symbolic algorithms scale better than explicit ones. For example, in hardware model checking symbolic algorithms enable going from millions of states to 10^{20} states and more [21, 22]. In contrast, in the context of PG solvers, symbolic algorithms have been only marginally explored. In this direction we just mention a symbolic implementation of `RE` [23, 24], which, however, has been done for different purposes and no benchmark comparison with the explicit version has been carried out. Other works close to this topic and worth mentioning are [25, 26], where a symbolic version of `SPM` has been theoretically studied but not implemented.

In [27, 28] a first broad investigation of the symbolic approach for solving PGs is provided. We implement four symbolic algorithms and compare their performances to the corresponding explicit versions for different classes of PGs [29]. Specifically, we implement in a new tool, called `SymPGSolver`[1], the symbolic versions of `RE`, `APT`, and two variants of `SPM`. The tool also allows to generate random games, as well as compare the performance of different symbolic algorithms.

Our analysis started from constrained random games [30]. The results show that on these games the explicit approach is better than the symbolic one, exhibiting a different behavior than the one showed in [30]. To gain a fuller understanding of the performances of the symbolic and the explicit algorithms, we have further tested the two approaches on structured games. Precisely, we have considered ladder games, clique games, as well as game models coming from

[1]The tool is available for download from https://github.com/antoniodistasio/sympgsolver

practical model-checking problems.

Ladder Games. In a ladder game, every node in P_i has priority i. In addition, each node $v \in \mathrm{P}$ has two successors: one in P_0 and one in P_1, which form a node pair. Every pair is connected to the next pair forming a ladder of pairs. Finally, the last pair is connected to the top. The parameter m specifies the number of node pairs. Formally, a ladder game of index m is $\mathcal{G} = (\mathrm{P}_0, \mathrm{P}_1, Mv, \mathsf{p})$ where $\mathrm{P}_0 = \{0, 2, \ldots, 2m-2\}$, $\mathrm{P}_1 = \{1, 3, \ldots, 2m-1\}$, $Mv = \{(v, w) | w \equiv_{2m} v + i$ for $i \in \{1, 2\}\}$, and $\mathsf{p}(v) = v \; mod \; 2$. Tables 1 and 2 reports the benchmarks.

m	SRE	SAPT	SSP	SSP2
1,000	0	0.00013	24.86	0.47
10,000	0.00009	0.00016	$\mathtt{abort}_T$	41.22
100,000	0.0001	0.00018	$\mathtt{abort}_T$	$\mathtt{abort}_T$
1,000,000	0.00012	0.00022	$\mathtt{abort}_T$	$\mathtt{abort}_T$
10,000,000	0.00015	0.00025	$\mathtt{abort}_T$	$\mathtt{abort}_T$

Table 1
Runtime executions of the symbolic algorithms on ladder games.

m	RE	APT	SPM
1,000	0.0007	0.0006	0.002
10,000	0.006	0.005	0.0017
100,000	0.057	0.054	0.18
1,000,000	0.59	0.56	1.84
10,000,000	6.31	5.02	20.83

Table 2
Runtime executions of the explicit algorithms on ladder games.

Benchmarks indicate that SRE and SAPT outperform their explicit versions, showing an excellent runtime execution even on fairly large instances. Indeed, while RE needs 6.31 seconds for games with index $m = 10M$, SRE takes just 0.00015 seconds. Tests also show that SSP and SSP2 have yet the worst performance.

Clique Games. Clique games are fully connected games without self-loops, where P_0 (*resp.*, P_1) contains the nodes with an even index (*resp., odd*) and each node $v \in \mathrm{P}$ has as priority the index of v. An important feature of the clique games is the high number of cycles, which may pose difficulties for certain algorithms. Formally, a clique game of index n is $\mathcal{G} = (\mathrm{P}_0, \mathrm{P}_1, Mv, \mathsf{p})$ where $\mathrm{P}_0 = \{0, 2, \ldots, n-2\}$, $\mathrm{P}_1 = \{1, 3, \ldots, n-1\}$, $Mv = \{(v, w) | v \neq w\}$, and $\mathsf{p}(v) = v$. Benchmarks on clique games are reported in Tables 3 and 4.

n	SRE	SAPT	SSP	SSP2
2,000	0.007	0.003	5.53	$\mathtt{abort}_T$
4,000	0.018	0.008	19.27	$\mathtt{abort}_T$
6,000	0.025	0.012	39.72	$\mathtt{abort}_T$
8,000	0.037	0.017	76.23	$\mathtt{abort}_T$

Table 3
Runtime executions of the symbolic algorithms on clique games

n	RE	APT	SPM
2,000	0.021	0.0105	0.0104
4,000	0.082	0.055	0.055
6,000	0.19	0.21	0.22
8,000	0.35	0.59	0.63

Table 4
Runtime executions of the explicit algorithms on clique games

The main result we obtain from our comparisons is that for random games, and even for constrained random games, explicit algorithms actually perform better than symbolic ones, most likely because BDDs do not offer any compression for random sets. The situation changes, however, for structured games, where symbolic algorithms sometimes outperform explicit algorithms. This is similar to what has been observed in the context of model checking [31].

n	Pr	Property	SRE	SAPT	SSP	SSP2	RE	APT	SPM	WS	DS
14,065	3	ND	0.00009	0.00006	3.30	0.0001	0.004	0.004	0.029	2	2
17,810	3	IORD1	0.0003	0.0005	$\texttt{abort}_T$	85.4	0.006	0.006	0.037	2	2
34,673	3	IORW	0.0006	0.0008	164.73	56.44	0.015	0.014	0.053	2	2
2,589,056	3	ND	0.0002	$\texttt{abort}_T$	$\texttt{abort}_T$	0.29	1.02	0.93	9.09	4	2
3,487,731	3	IORD1	$\texttt{abort}_T$	$\texttt{abort}_T$	$\texttt{abort}_T$	$\texttt{abort}_T$	1.81	1.4	17.45	4	2
6,823,296	3	IORW	0.3	$\texttt{abort}_T$	$\texttt{abort}_T$	$\texttt{abort}_T$	3.87	3.13	22.26	4	2

Table 5
SWP (Sliding Window Protocol)

n	Pr	Property	SRE	SAPT	SSP	SSP2	RE	APT	SPM	DS
81,920	3	ND	0.00002	31.69	1.37	0.0016	0.031	0.034	0.22	2
88,833	3	IORD1	0.0027	0.003	$\texttt{abort}_T$	$\texttt{abort}_T$	0.036	0.0038	0.27	2
170,752	3	IORW	14.37	98.4	$\texttt{abort}_T$	$\texttt{abort}_T$	0.07	0.07	0.47	2
289,297	3	ND	0.0001	154.89	12.3	0.0058	0.13	0.12	1.34	4
308,737	3	IORD1	0.0088	0.009	$\texttt{abort}_T$	$\texttt{abort}_T$	0.14	0.13	1.37	4
607,753	3	IORW	43.7	$\texttt{abort}_T$	$\texttt{abort}_T$	$\texttt{abort}_T$	0.29	0.27	2.06	4

Table 6
OP (Onebit Protocol)

n	Pr	Property	SRE	SAPT	SSP	SSP2	RE	APT	SPM	DS
328	1	ND	0.00002	0.002	0.005	0.00002	0.0001	0.0001	0.0004	2
308	1	safety	0.00002	0.003	0.028	0.00002	0.0001	0.0001	0.0004	2
655	3	liveness	0.00008	0.0001	5.52	0.09	0.0003	0.0002	0.001	2
51.220	1	safety	0.0001	1.48	32.14	0.00002	0.01	0.01	0.09	4
53.638	1	ND	0.0001	0.2	4.67	0.0001	0.017	0.015	0.07	4
107,275	3	liveness	0.005	0.001	$\texttt{abort}_T$	$\texttt{abort}_T$	0.03	0.03	0.18	4

Table 7
Lift (Lifting Truck)

Finally, we evaluate the symbolic and explicit approaches on some practical model checking problems as in [32]. Specifically, we use models coming from: the Sliding Window Protocol (SWP) with window size (WS) of 2 and 4 (WS represents the boundary of the total number of packets to be acknowledged by the receiver), the Onebit Protocol (OP), and the Lifting Truck (Lift). The properties we check on these models concern: absence of deadlock (ND), a message of a certain type (d1) is received infinitely often (IORD1), if there are infinitely many read steps then there are infinitely many write steps (IORW), liveness, and safety. Note that, in all benchmarks, data size (DS) denotes the number of messages.

As we can see, by comparing Tables 5, 6, and 7, the experiments indicate more nuanced relationship between the symbolic and explicit approaches. Indeed, they show a different behavior depending on the protocol and the property we are checking. Overall, we note that

SRE outperforms the other symbolic algorithms in all protocols, although the advantage over RE is discontinued. Specifically, SRE is the best performing in checking absence of deadlock in all three protocols, but for IORD1 in the SWP protocol with $WS = 2$, or for IORW in the OP protocol, RE exhibits a significant advantage. Differently, SAPT and SSP2 show better performances on a smaller number of properties. Moreover, the results highlights that SSP exhibits the worst performances in all protocols and properties.

We take this as an important development because it suggests a methodological weakness in this field of investigation, due to the excessive reliance on random benchmarks. We believe that, in evaluating algorithms for PG solving, it would be useful to have real benchmarks and not only random benchmarks, as the common practice has been. This would lead to a deeper understanding of the relative merits of PG solving algorithms, both explicit and symbolic.

Acknowledgments

We thank our co-authors on the publications mentioned in this communication: Aniello Murano and Moshe Y. Vardi. This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), by the EU ICT-48 2020 project TAILOR (No. 952215), and by the PRIN project RIPER (No. 20203FFYLK).

References

[1] E. Emerson, C. Jutla, Tree Automata, μ-Calculus and Determinacy, in: 32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991, 1991, pp. 368–377.

[2] O. Kupferman, M. Y. Vardi, Weak Alternating Automata and Tree Automata Emptiness, in: Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998, 1998, pp. 224–233.

[3] E. Clarke, E. Emerson, Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic, in: Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981, LNCS 131, 1981, pp. 52–71.

[4] P. Cermák, A. Lomuscio, A. Murano, Verifying and synthesising multi-agent systems against one-goal strategy logic specifications, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA, 2015, pp. 2038–2044.

[5] O. Kupferman, M. Vardi, P. Wolper, An Automata Theoretic Approach to Branching-Time Model Checking, J. ACM 47 (2000) 312–360.

[6] T. Wilke, Alternating Tree Automata, Parity Games, and Modal μ-Calculus, Bulletin of the Belgian Mathematical Society Simon Stevin 8 (2001) 359.

[7] E. Clarke, O. Grumberg, D. Peled, Model Checking., MIT Press, 2002.

[8] J. Queille, J. Sifakis, Specification and Verification of Concurrent Programs in Cesar, in: International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings, LNCS 137, 1982, pp. 337–351.

[9] O.Kupferman, M.Vardi, P.Wolper, Module Checking, Information and Computation. 164 (2001) 322–344.
[10] W. Thomas, Facets of Synthesis: Revisiting Church's Problem, in: Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009., LNCS 5504, 2009, pp. 1–14.
[11] B. Aminof, O. Kupferman, A. Murano, Improved Model Checking of Hierarchical Systems, Inf. Comput. 210 (2012) 68–86.
[12] F. Mogavero, A. Murano, L. Sorrentino, On Promptness in Parity Games, in: Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013., LNCS 8312, 2013, pp. 601–618.
[13] D. Kozen, Results on the Propositional μ-Calculus, Theoretical Computer Science 27 (1983) 333–354.
[14] M. Jurdzinski, Deciding the Winner in Parity Games is in UP $\cap$ co-Up, Inf. Process. Lett. 68 (1998) 119–124.
[15] W. Zielonka, Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees, Theor. Comput. Sci. 200 (1998) 135–183.
[16] M. Jurdzinski,Small Progress Measures for Solving Parity Games, in: STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings, LNCS 1770, 2000, pp. 290–301.
[17] A. Di Stasio, A. Murano, G. Perelli, M. Y. Vardi, Solving parity games using an automata-based algorithm, in: Implementation and Application of Automata - 21st International Conference, CIAA 2016, Seoul, South Korea, July 19-22, 2016., 2016, pp. 64–76.
[18] C. S. Calude, S. Jain, B. Khoussainov, W. Li, F. Stephan, Deciding parity games in quasipolynomial time, in: STOC 2017, 2017, pp. 252–263.
[19] M. Jurdzinski, M. Paterson, U. Zwick, A deterministic subexponential algorithm for solving parity games, in: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006, ACM Press, 2006, pp. 117–123.
[20] R. E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Trans. Comput. (1986) 677–691.
[21] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang, Symbolic model checking: 10ˆ20 states and beyond, in: Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990, 1990, pp. 428–439.
[22] K. L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.
[23] G. Kant, J. van de Pol, Generating and solving symbolic parity games, in: Proceedings 3rd Workshop on GRAPH Inspection and Traversal Engineering, GRAPHITE 2014, Grenoble, France, 5th April 2014, 2014, pp. 2–14.
[24] M. Bakera, S. Edelkamp, P. Kissmann, C. D. Renner, Solving μ-calculus parity games by symbolic planning, in: Model Checking and Artificial Intelligence, 5th International Workshop, MoChArt 2008, Patras, Greece, July 21, 2008., 2008, pp. 15–33.
[25] D. Bustan, O. Kupferman, M. Y. Vardi, A measured collapse of the modal μ-calculus alternation hierarchy, in: STACS 2004, 21st Annual Symposium on Theoretical Aspects of

Computer Science, Montpellier, France, March 25-27, 2004, Proceedings, 2004, pp. 522–533.
[26] K. Chatterjee, W. Dvorák, M. Henzinger, V. Loitzenbauer, Improved set-based symbolic algorithms for parity games, in: 26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden, 2017, pp. 18:1–18:21.
[27] A. Di Stasio, A. Murano, M. Y. Vardi, Solving parity games: Explicit vs symbolic, in: Implementation and Application of Automata - 23rd International Conference, CIAA 2018, Charlottetown, PE, Canada, July 30 - August 2, 2018, Proceedings, 2018, pp. 159–172.
[28] A. D. Stasio, Reasoning about LTL Synthesis over finite and infinite games, Ph.D. thesis, University of Naples Federico II, Italy, 2018.
[29] T. van Dijk, Oink: An implementation and evaluation of modern parity game solvers, in: Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, LNCS 10805, Springer, 2018, pp. 291–308.
[30] D. Tabakov, Evaluation of Explicit and Symbolic Automata-Theoretic Algorithm, Master's thesis, Rice University, 2005.
[31] C. Eisner, D. A. Peled, Comparing symbolic and explicit model checking of a software system, in: Model Checking of Software, 9th International SPIN Workshop, Grenoble, France, April 11-13, 2002, Proceedings, 2002, pp. 230–239.
[32] J. A. Keiren, Benchmarks for parity games, in: FSEN 2015, 2015, pp. 127–142.