Solving Parity Games: Explicit vs Symbolic

Antonio Di Stasio[1] Aniello Murano[1] Moshe Vardi[2]

[1]University of Napoli "Federico II"

[2]Rice University

CIAA, 2018

Importance of Parity Games

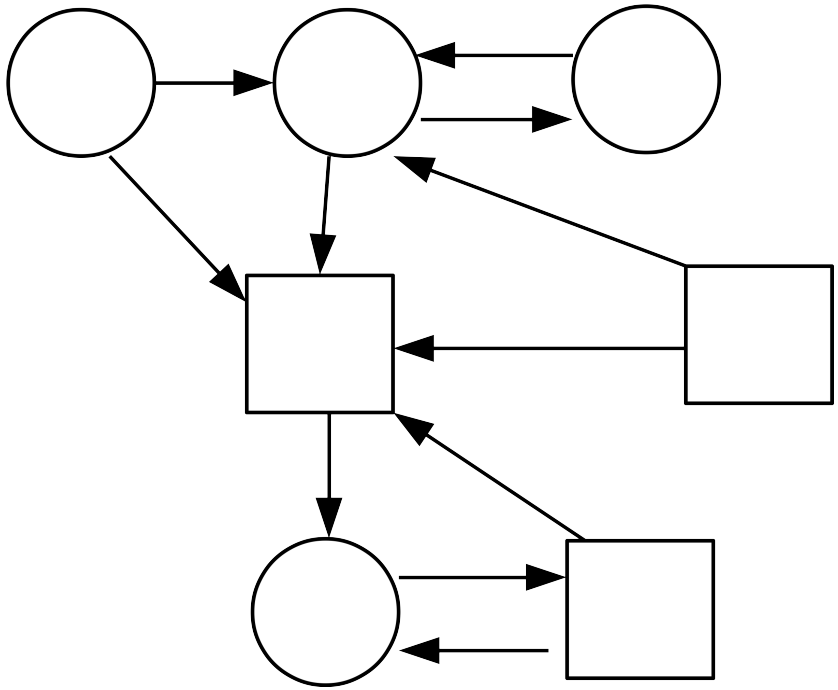
- A powerful formalism to synthesize and verify reactive systems.
- Correspond to the model-checking problem for the modal mu-calculus.
- Automata theory, e.g., complementation and determination.
- They express important system requirements (safety, liveness, etc.).

However,

- Finding a winning strategy in Parity Games is in UP $\cap$ Co-UP.
- Deciding whether a polynomial time solution exists is a long-standing open question.

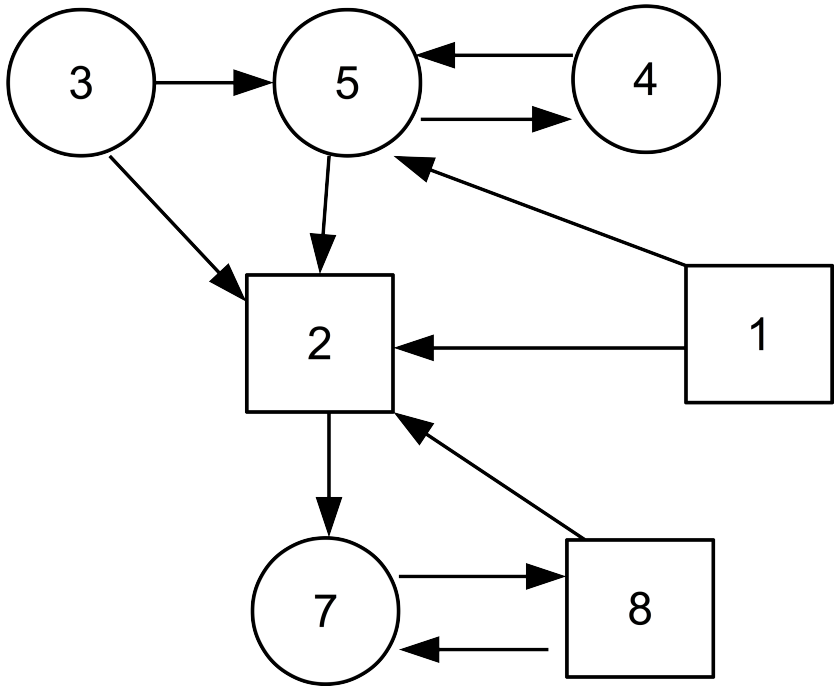
Parity Games

- Parity Games are infinite two-player games:
  1. Player 0 ○
  2. Player 1 □
- They are played on directed graphs.



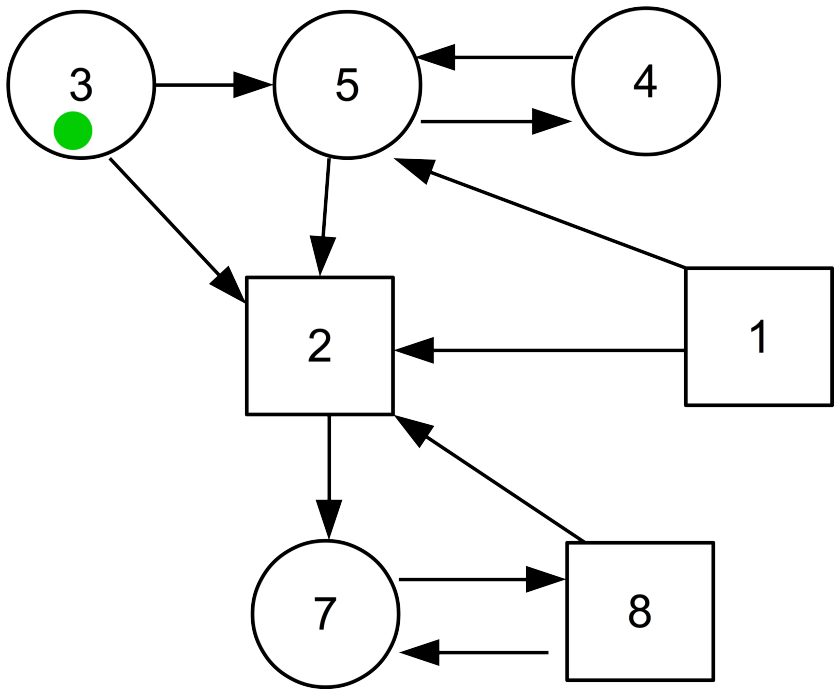
Parity Games

- Parity Games are infinite two-player games:
  1. Player 0 ○
  2. Player 1 □
- They are played on directed graphs.
- Each node is labeled by a priority, i.e., a natural number.



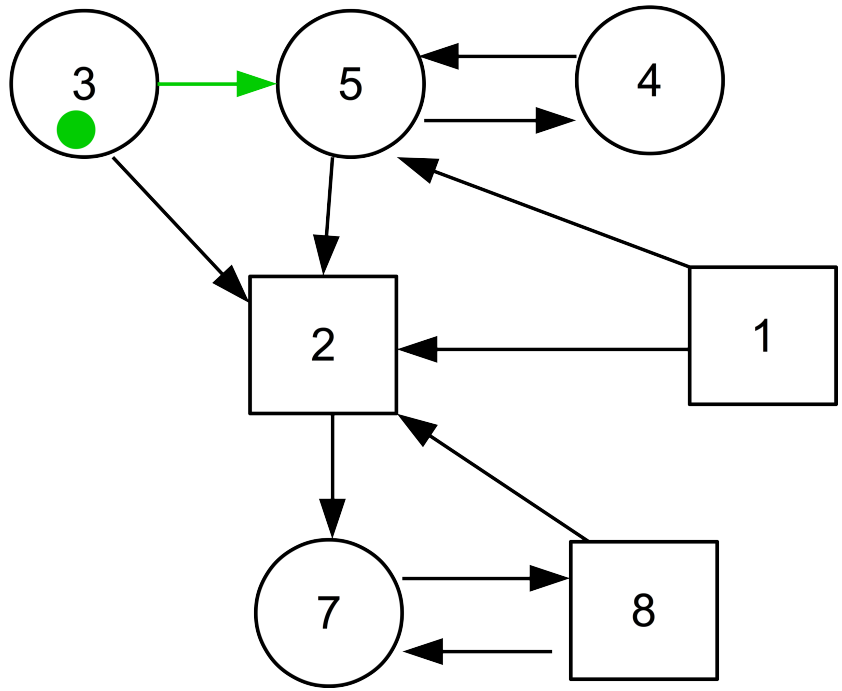
Parity Games

- The players move in turn a token along graph's edges
- The **play** induces an infinite path in which:
  - Player 0 wins the play if the smallest priority visited infinitely often is even;
  - otherwise, Player 1 wins the play.



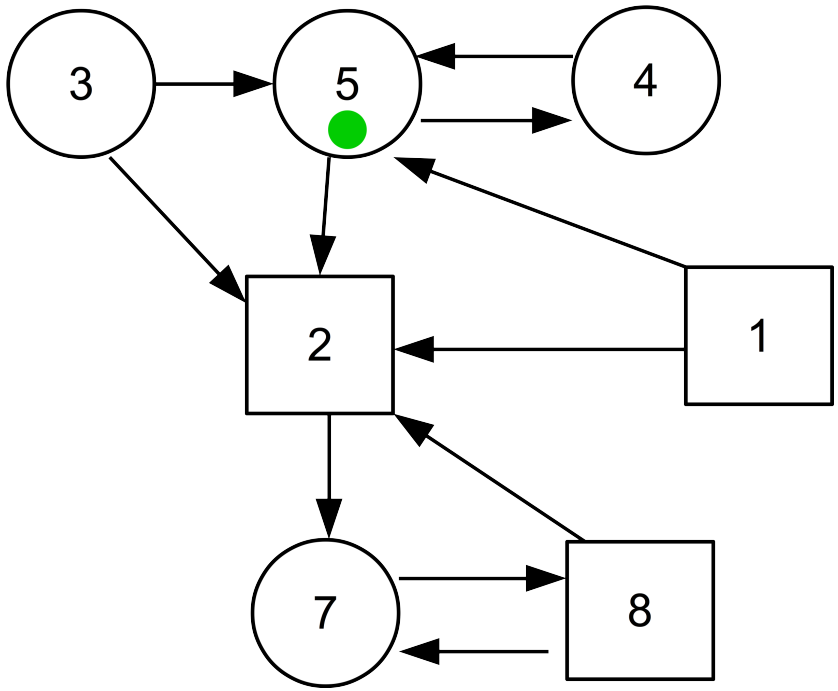
Parity Games

- The players move in turn a token along graph's edges
- The **play** induces an infinite path in which:
  - Player 0 wins the play if the smallest priority visited infinitely often is even;
  - otherwise, Player 1 wins the play.



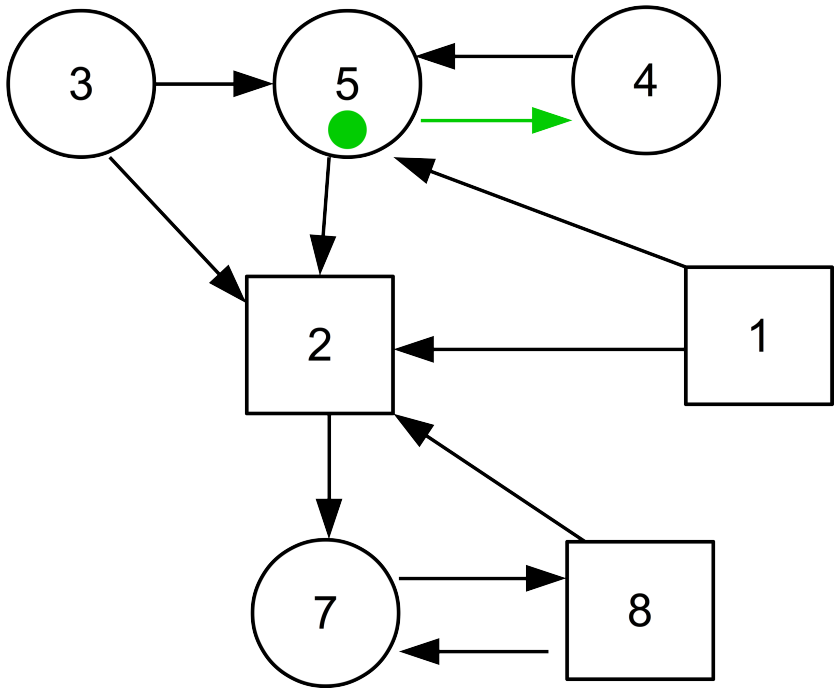
Parity Games

- The players move in turn a token along graph's edges
- The **play** induces an infinite path in which:
  - Player 0 wins the play if the smallest priority visited infinitely often is even;
  - otherwise, Player 1 wins the play.



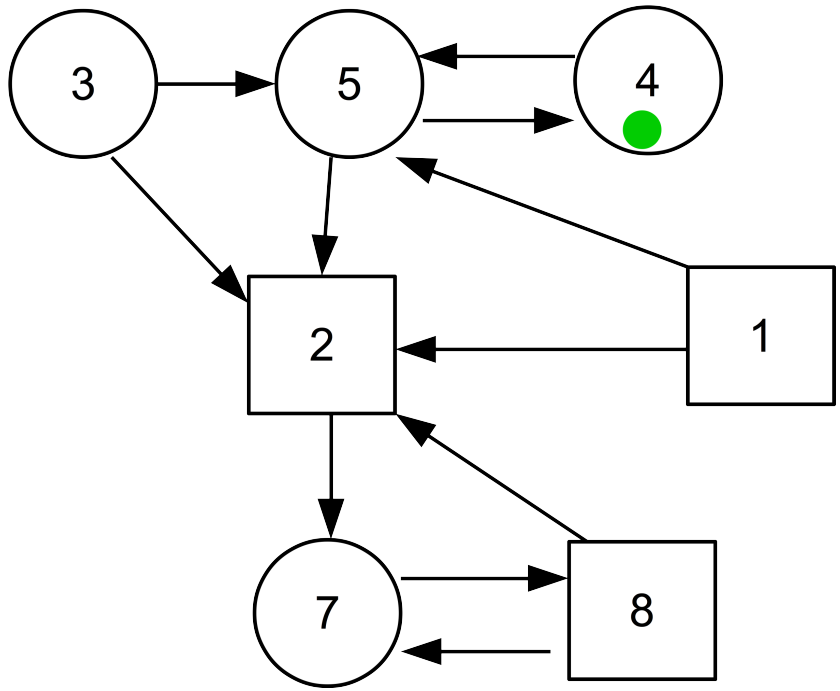
Parity Games

- The players move in turn a token along graph's edges
- The **play** induces an infinite path in which:
  - Player 0 wins the play if the smallest priority visited infinitely often is even;
  - otherwise, Player 1 wins the play.



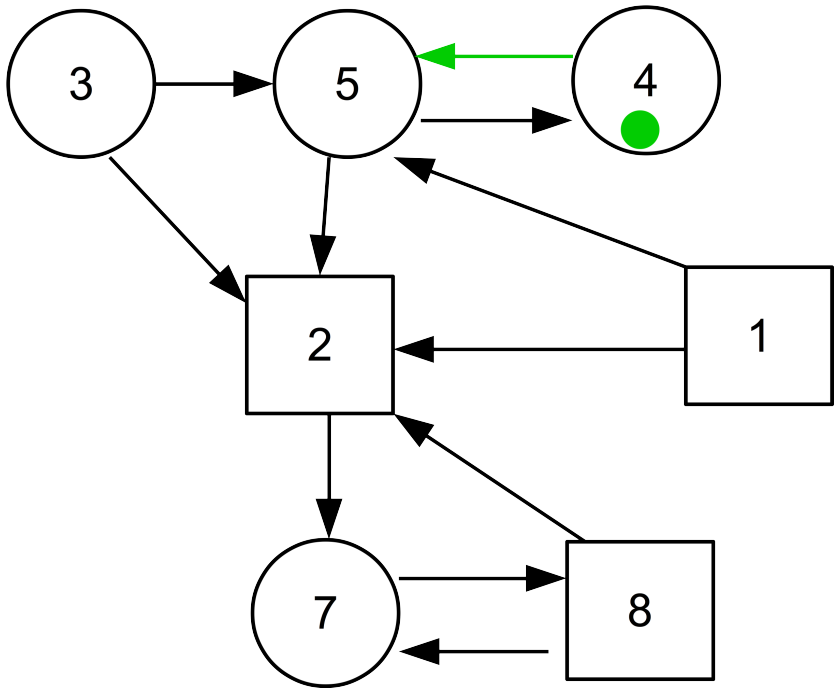
Parity Games

- The players move in turn a token along graph's edges
- The **play** induces an infinite path in which:
  - Player 0 wins the play if the smallest priority visited infinitely often is even;
  - otherwise, Player 1 wins the play.



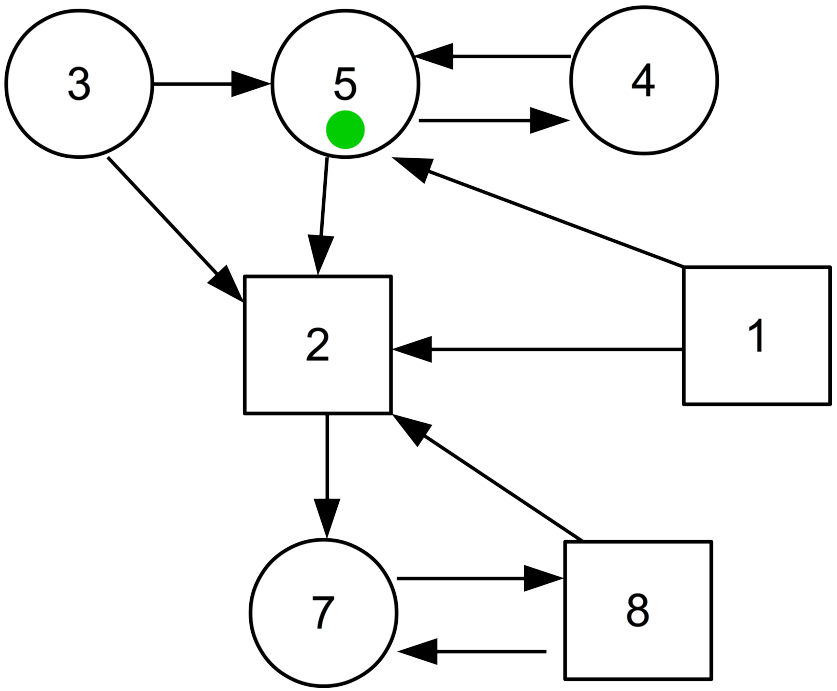
Parity Games

- The players move in turn a token along graph's edges
- The **play** induces an infinite path in which:
  - Player 0 wins the play if the smallest priority visited infinitely often is even;
  - otherwise, Player 1 wins the play.



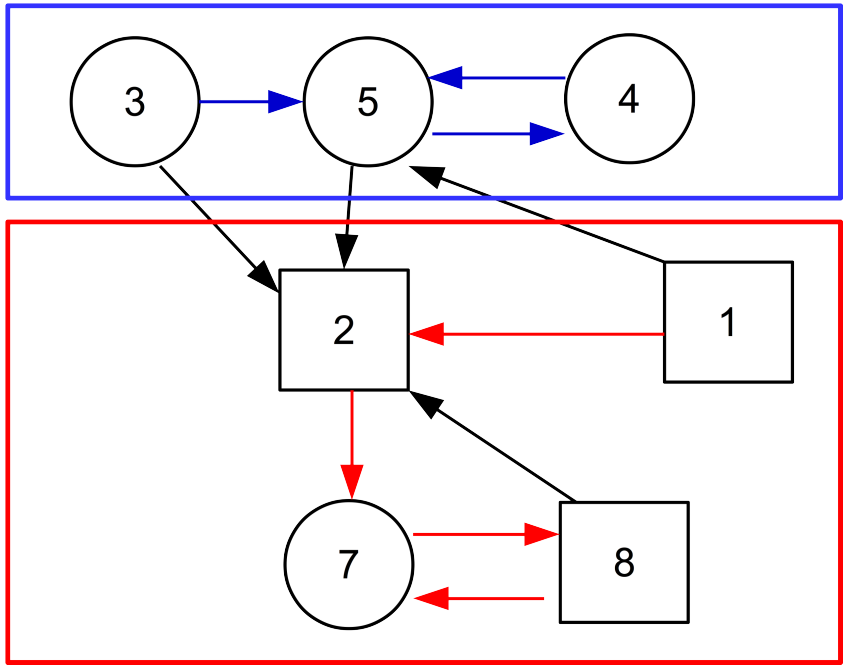
Parity Games

- The players move in turn a token along graph's edges
- The **play** induces an infinite path in which:
  - Player 0 wins the play if the smallest priority visited infinitely often is even;
  - otherwise, Player 1 wins the play.



Parity Games

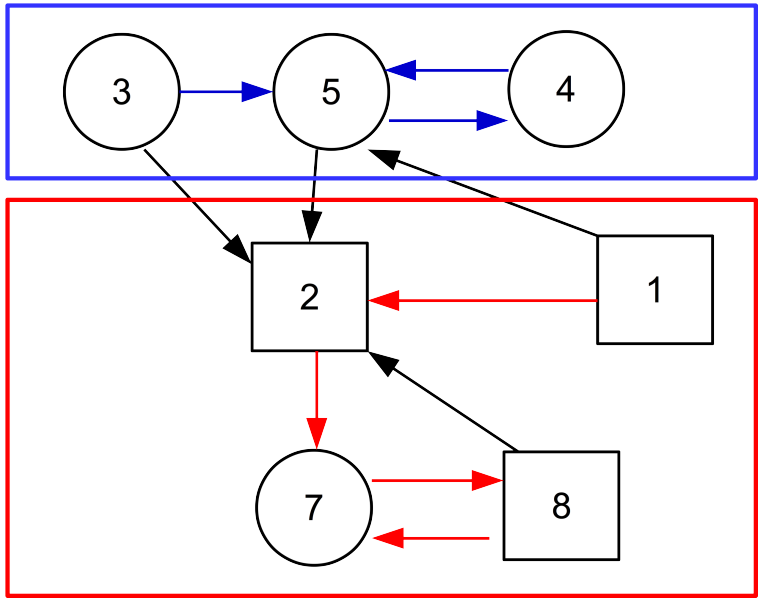
- The players move in turn a token along graph's edges
- The **play** induces an infinite path in which:
  - Player 0 wins the play if the smallest priority visited infinitely often is even;
  - otherwise, Player 1 wins the play.
- Solving parity games means determining the winner at each node.



Parity Games - Determinacy

A game is determined if for every node v either

- Player 0 can ensure winning from v or
- Player 1 can ensure winning from v.



Theorem ([Martin, 1975])

Parity games are determined

Algorithms for Parity Games

Main Algorithms	Computational Complexity [1]
Recursive (Rec)	$O(e \cdot n^k)$
Small Progress Measures (SPM)	$O(k \cdot e \cdot (\frac{n}{k})^{\frac{k}{2}})$
APT	$O(n^k)$
Strategy Improvement (SI)	$O(2^e \cdot n \cdot e)$
Dominion Decomposition (DD)	$O(n^{\sqrt{n}})$
Big Step (BS)	$O(e \cdot n^{\frac{1}{3}k})$
Quasi-polynomial (QPT)	$O(n^{\log k+6})$

Table: Parity algorithms along with their computational complexities.

[1] n = nodes, e = number of edges, k = priorities

Algorithms for Parity Games

Main Algorithms	Computational Complexity [2]
Recursive (Rec)	$O(e \cdot n^k)$
Small Progress Measures (SPM)	$O(k \cdot e \cdot (\frac{n}{k})^{\frac{k}{2}})$
APT	$O(n^k)$
Strategy Improvement (SI)	$O(2^e \cdot n \cdot e)$
Dominion Decomposition (DD)	$O(n^{\sqrt{n}})$
Big Step (BS)	$O(e \cdot n^{\frac{1}{3}k})$
Quasi-polynomial (QPT)	$O(n^{\log k+6})$

Table: Parity algorithms along with their computational complexities.

[2] n = nodes, e = number of edges, k = priorities

APT Algorithm

- `APT` was introduced by Orna Kupferman and Moshe Vardi in "Weak Alternating Automata and Tree Automata Emptiness" [KV98].
- `APT` solves parity games via the emptiness of alternating parity automata.
- It was just sketched but not spelled out in detail. Moreover, no proof correctness was provided.
- Two major gaps were filled in "Solving Parity Games Using Automata Based Algorithm" by Di Stasio, Murano, Perelli, Vardi CIAA 2016.

APT Algorithm - Benchmarks

- APT was implemented in the platforms PGSolver and Oink.
- By means of benchmarks, we showed that `APT` is the best performing algorithm for solving parity games when:
  - the number of priorities is much smaller than the number of nodes in the graph;
  - the number of priorities is logarithmic in the number of the nodes.

Small Progress Measures (SPM)

Idea..

A progress measure based on a ranking function, where:

- the ranking function assigns to each node a vector of counters collecting the number of times Player 1 can force a play to visit an odd priority until a lower even priority is seen.
- the progress measure is computable by updating the rank of a node according to the ranks of its successors in a fixpoint iteartion.

Recursive Algorithm (RE)

- Zielonka's recursive algorithm is based on attractor computation.

Definition *Attractor*

Given a set of nodes U, the attractor of U for a Player i represents those nodes that i can force the play toward.

Despite its relatively bad theoretical complexity is known to outperform other algorithms in practice.

Symbolic Representation of a Parity Game

- All main algorithms for solving parity games 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.
- Symbolic algorithms, based on the manipulation of Boolean formulas, are an efficient way to deal with extremely large graphs (e.g., symbolic model checking).

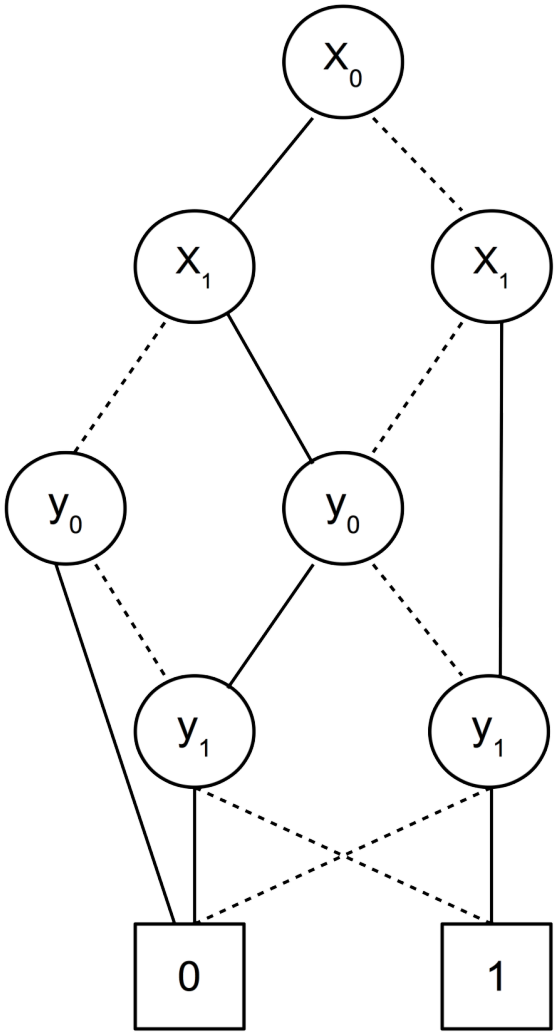
Symbolic Representation of a Parity Game

Implementation

A Symbolic Parity Game can be implemented using

- Binary Decision Diagrams (BDDs) to represent nodes and transitions,
- Algebraic Decision Diagrams (ADDs) to represent the priority function.

Binary Decision Diagrams



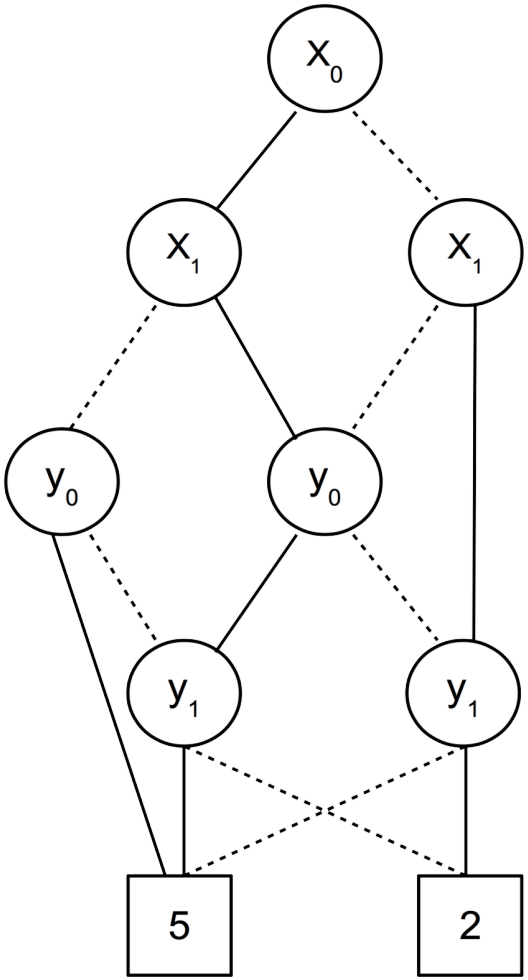
BDDs represent a function $f : \mathbb{B}^d \to \mathbb{B}$.

BDDs provide efficient implementations for

- $\neg, \vee, \wedge$
- $\forall, \exists$
- $\equiv$

Widely used for computing fixpoints.

Algebraic Decision Diagrams



ADDs represent a function $f : \mathbb{B}^d \to \mathbb{R}$.

ADDs provide efficient implementations for

- $-, +, \times$
- max(f,g), min(f,g)

Implementation - Symbolic RE and APT

- The Recursive algorithm (RE) and APT are *set-based algorithms*, that is, based on set operations like union, intersection, complementation, and inclusion.
- They can be easily rephrased symbolically by using BDDs to represent the operations they make use.
- Moreover, they do not need to manipulate ADDs.

Implementation - Symbolic Small Progress Measures

There are two different symbolic implementations of Small Progress Measures:

1. In "A Measured Collapse of The Modal μ-Calculus Alternation Hierarchyby", by Bustan, Kupferman and Vardi (SSP) the algorithm computes the progress measure by manipulating ADDs.
2. In "Improved Set-Based Symbolic Algorithms for Parity Games", by Chatterjee, Dvorak, Henzinger and Loitzenbauer (SSP2), it is provided a set-based symbolic version of SPM, where the ranking function is encoded by using sets of nodes, and the progress measure is computed by using BDDs.

Benchmarks

- The symbolic algorithms have been implemented in a tool called `SymPGSolver` (Symbolic Parity Games Solver), implemented in C++.
- `SymPGSolver` uses the CUDD package as the underlying BDD and ADD library.
- We have compared the performance of the explicit algorithms `RE`, `APT`, and `SPM` implemented in *Oink*, developed by Tom van Dijk, with their corresponding symbolic versions `SRE`, `SAPT`, `SSP` and `SSP2`.

Benchmarks

We have used different classes of parity games:

- Random games with linear structures, where
  - exists a transition from v_i to v_j, if and only if $|i - j| \leq d$, where d fixed parameter.
- Ladder Games;
- Clique games;
- Games corresponding to practical model checking problems:
  - the Sliding Window Protocol, with window size (WS) of 2 and 4;
  - the Onebit Protocol;
  - the Lifting Truck.

Benchmarks - Random with linear structure

n	2 Pr SRE	SAPT	SSP	SSP2
1,000	0.04	0.03	29.89	0,95
2,000	0.14	0.12	128.06	2,87
3,000	0.25	0.23	$\texttt{abort}_T$	10,15
4,000	0.33	0.30	$\texttt{abort}_T$	32,42
7,000	0.79	0.73	$\texttt{abort}_T$	$\texttt{abort}_T$
10,000	1.16	1.12	$\texttt{abort}_T$	$\texttt{abort}_T$
20,000	2.78	3.10	$\texttt{abort}_T$	$\texttt{abort}_T$
100,000	19.21	24.4	$\texttt{abort}_T$	$\texttt{abort}_T$

Table: Runtime executions of the symbolic algorithms.

n	2 Pr RE	APT	SPM
1,000	0.0008	0.0006	0.0043
2,000	0.0015	0.0012	0.0084
3,000	0.0023	0.0017	0.012
4,000	0.0031	0.0022	0.016
7,000	0.0051	0.0039	0.025
10,000	0.0065	0.0057	0.035
20,000	0.013	0.011	0.078
100,000	0.094	0.081	0.44

Table: Runtime executions of the explicit algorithms.

Benchmarks - Ladder Games

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

Table: Runtime executions of the symbolic algorithms.

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: Runtime executions of the explicit algorithms.

Benchmarks - Clique Games

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

Table: Runtime executions of the symbolic algorithms.

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: Runtime executions of the explicit algorithms.

Benchmarks - Protocols

The properties we have checked on the model checking problems concern:

- absence of deadlock (ND);
- a message of a certain type(d1) is received infinitely often (IORD1);
- if there are infinitely often many read steps then there are infinitely write steps (IORW);
- liveness and safety.

Benchmarks - Protocols(1)

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: SWP (Sliding Window Protocol)

Benchmarks - Protocols(2)

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: OP (Onebit Protocol)

Benchmarks - Protocols(3)

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: Lift (Lifting Truck)

Conclusion and Future Work

We have:

- compared for the first time the performances of different symbolic and explicit versions of classic algorithms to solve parity games.
- implemented in a tool called `SymPGSolver` the symbolic versions of Recursive, APT, and Small Progress Measures algorithms.
- By means of benchmarks, shown that:
  - the explicit approach is better than the symbolic one for random games, even for constrained random games;
  - in several cases, symbolic algorithms outperform the explicit ones for structurated games.

Future work:

- Consider real scenarios to deeper understanding of the relative merits of parity games solving algorithms;
- Apply optimization techniques.