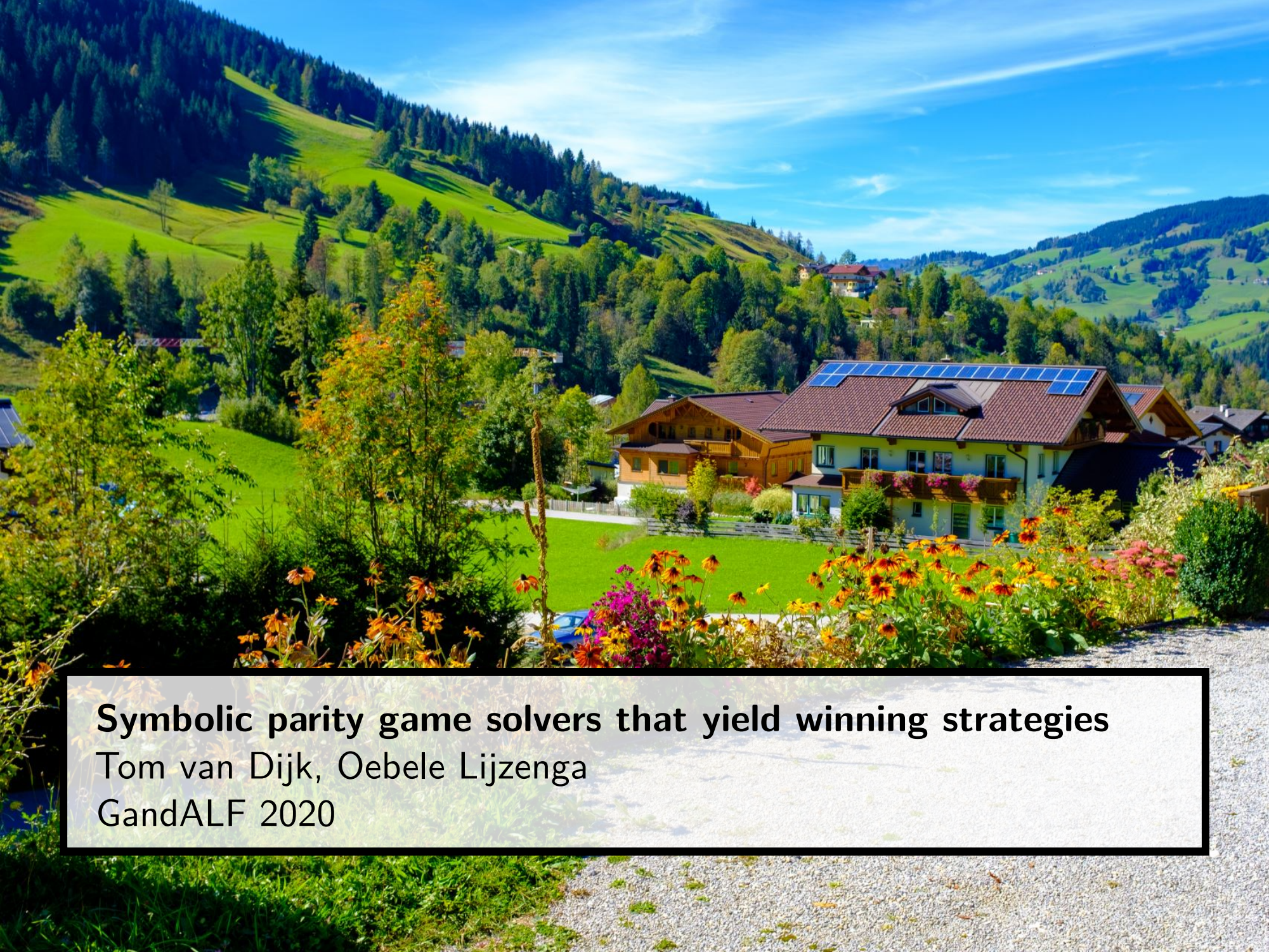


Symbolic parity game solvers that yield winning strategies

Tom van Dijk, Oebele Lijzenga

GandALF 2020

AUTHORS

Oebele Lijzenga

Tom van Dijk

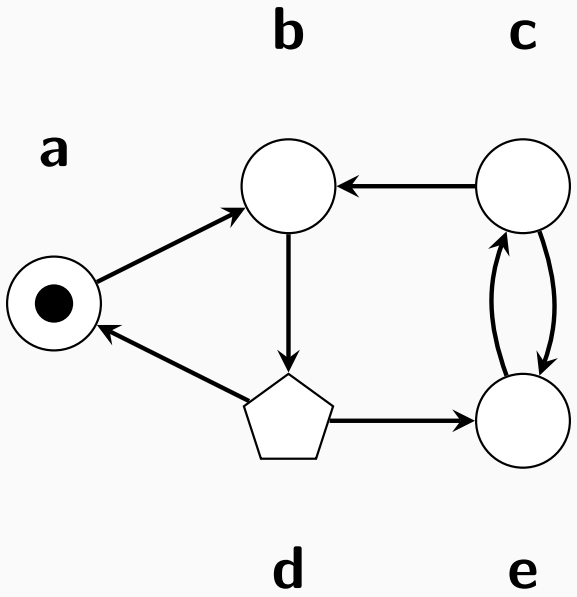
OVERVIEW

TL; DR

- Solving parity games symbolically
  - Symbolically means: using binary decision diagrams
- This paper: two symbolic algorithms that give winning strategies
  - "nested fixpoint" parity game algorithms are
    - extremely fast for explicit solving
    - naturally symbolic due to their set-based nature
  - two recent "nested fixpoint" algorithms yield strategies
- Context: towards symbolic LTL synthesis via parity games

PARITY GAMES

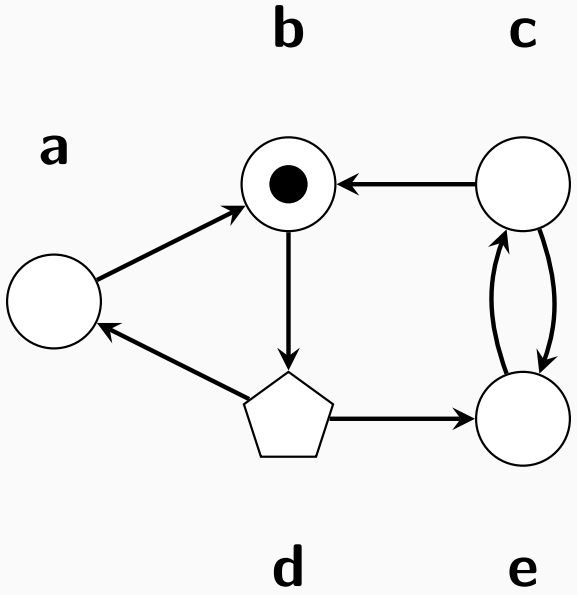
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a**

PARITY GAMES

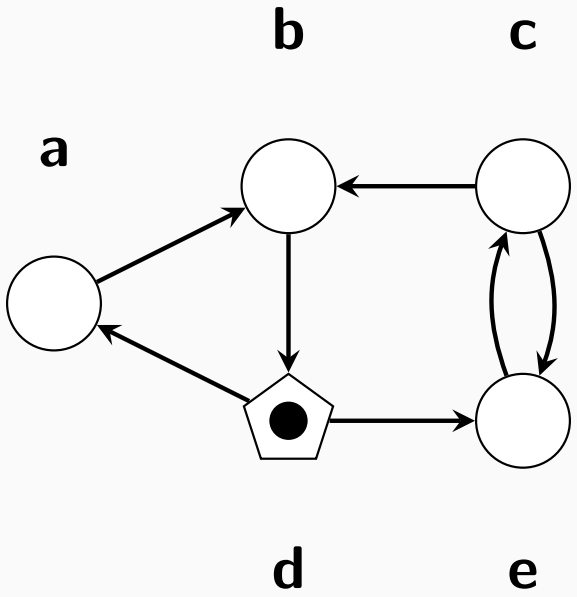
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b**

PARITY GAMES

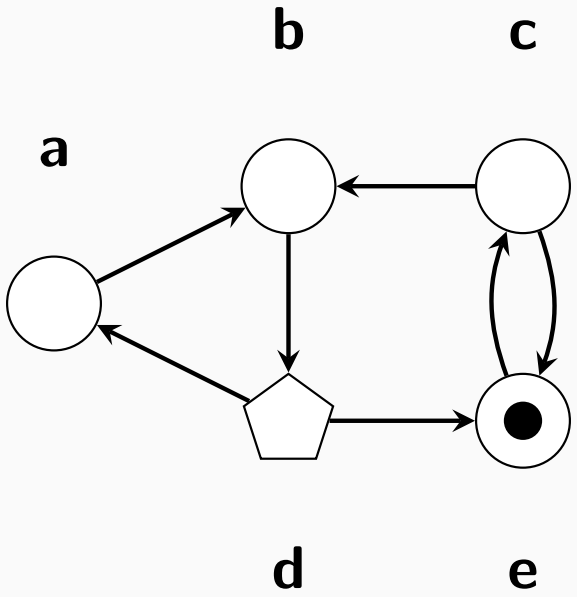
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d**

PARITY GAMES

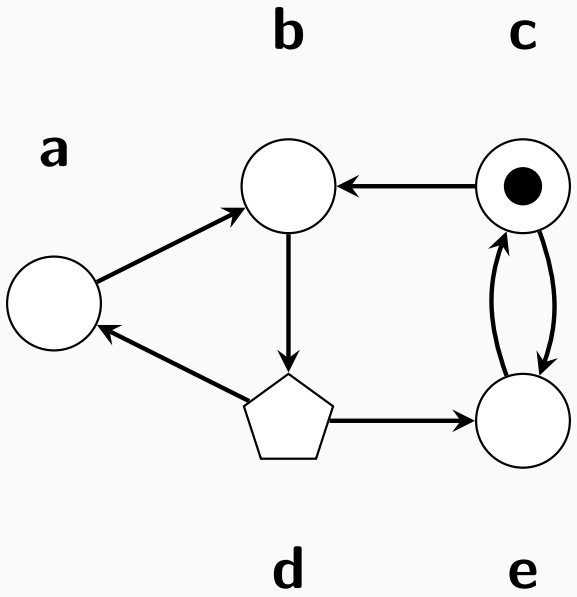
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e**

PARITY GAMES

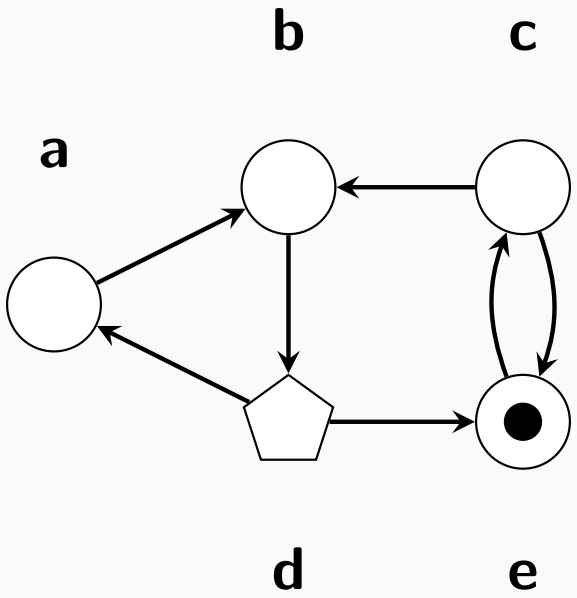
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c**

PARITY GAMES

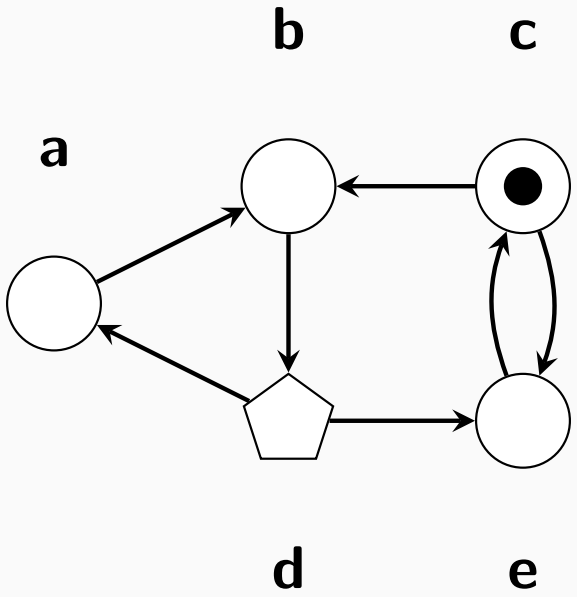
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e**

PARITY GAMES

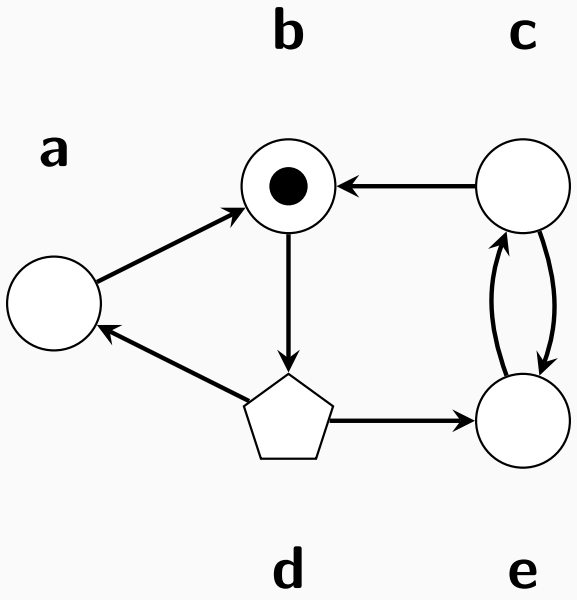
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c**

PARITY GAMES

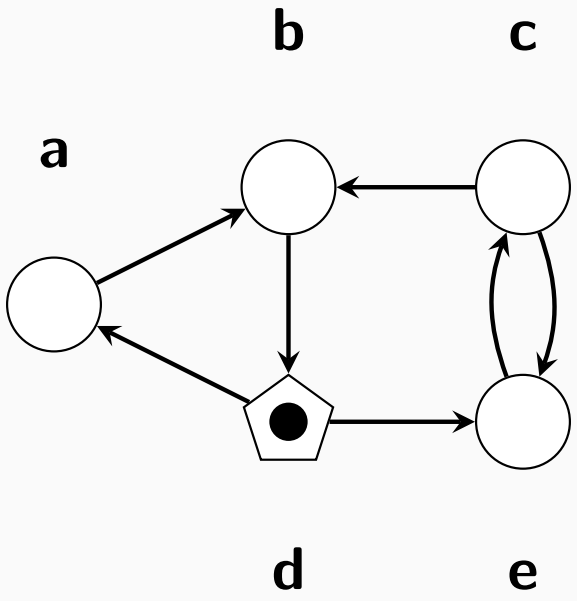
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b**

PARITY GAMES

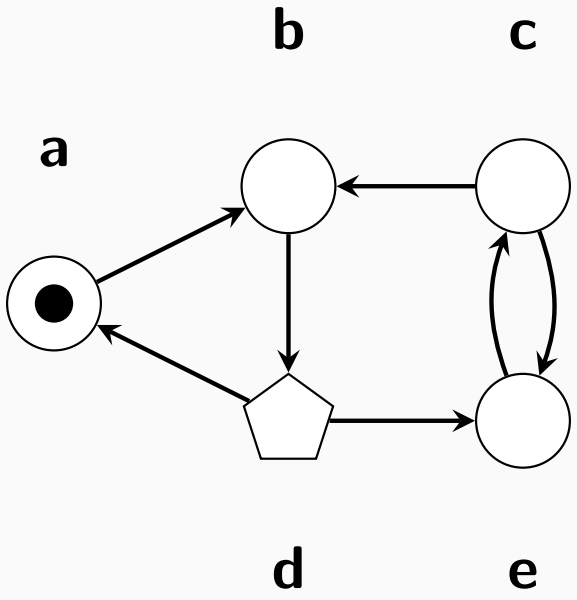
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b d**

PARITY GAMES

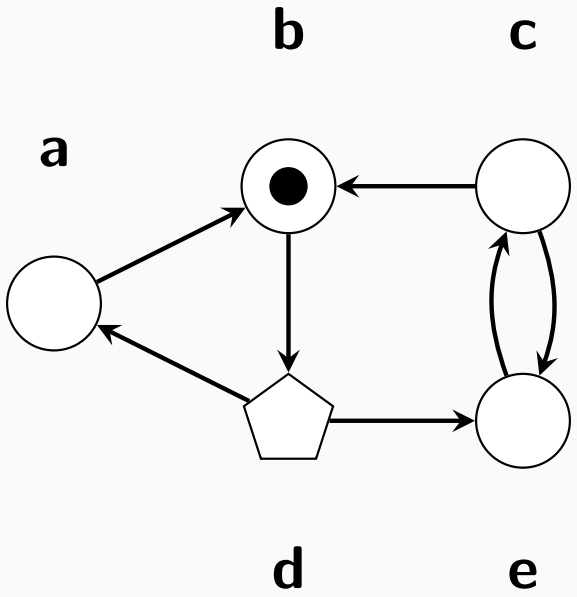
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b d a**

PARITY GAMES

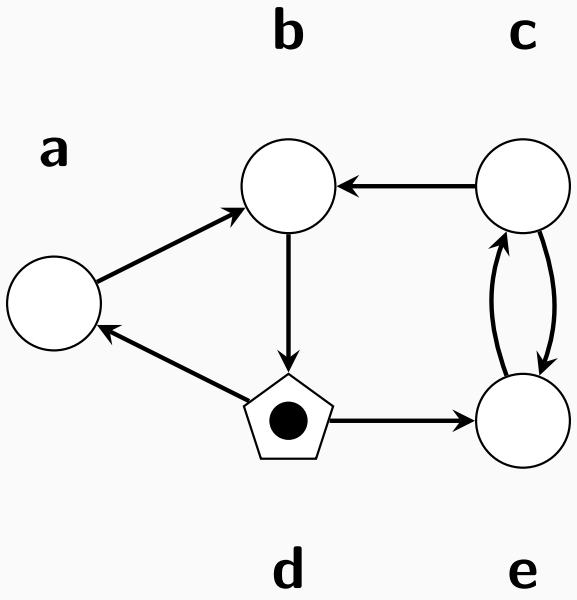
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b d a b**

PARITY GAMES

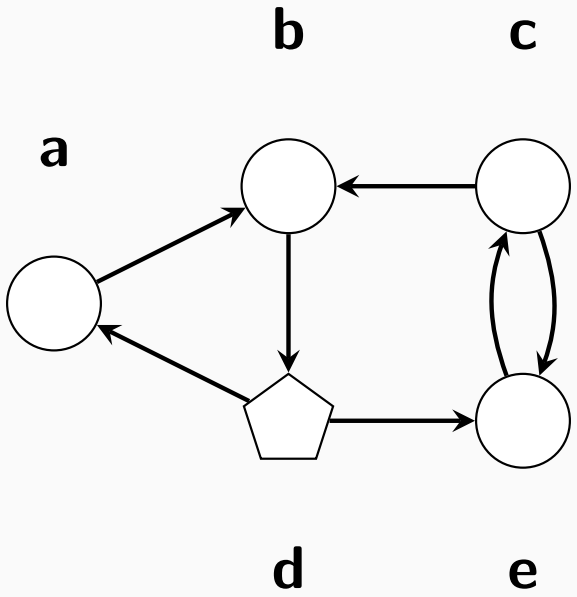
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b** $(\mathbf{d\ a\ b})^{\omega}$

PARITY GAMES

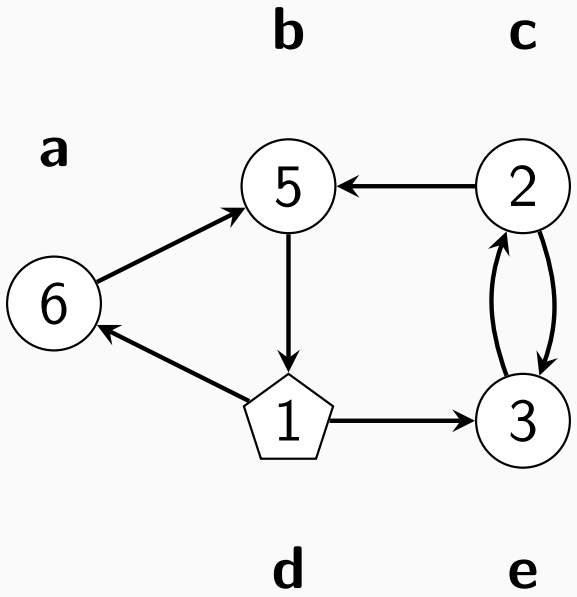
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



Who wins a play?

PARITY GAMES

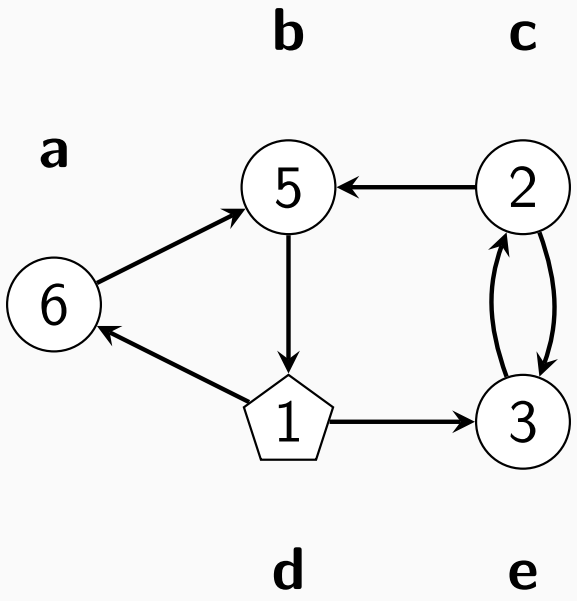
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



- Each vertex has a priority $\{0, 1, 2, \ldots, d\}$
- Player Even wins if the highest priority seen infinitely often is even

PARITY GAMES

- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor

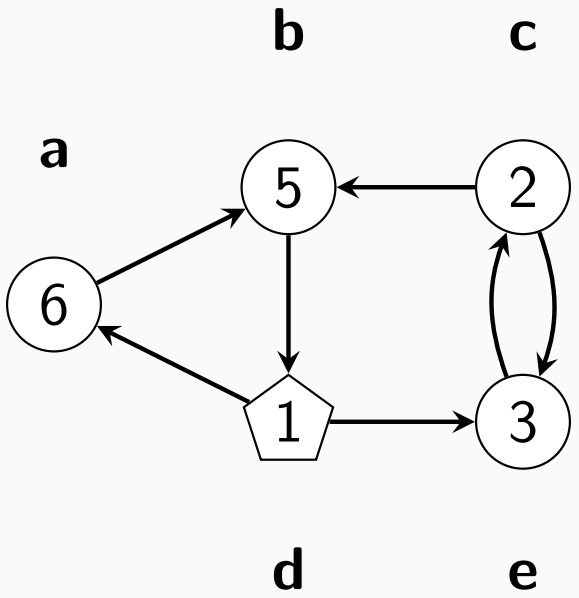


The play π: 6 5 1 3 2 3 2 5 $(1\ 6\ 5)^{\omega}$

Who wins this play?

PARITY GAMES

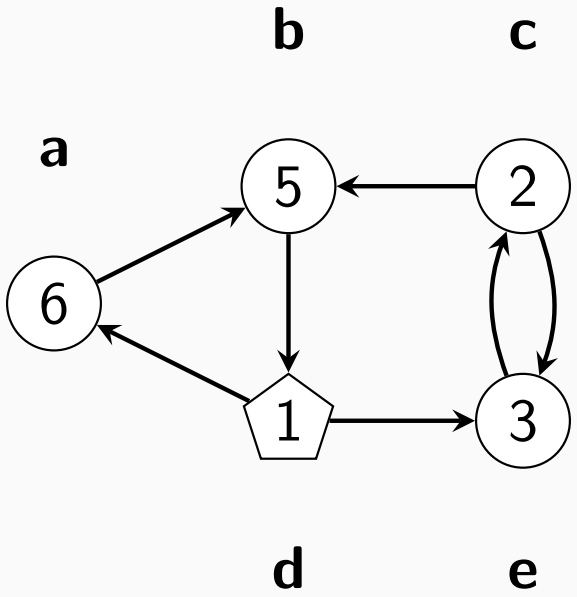
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



How do we determine who wins a vertex?

PARITY GAMES

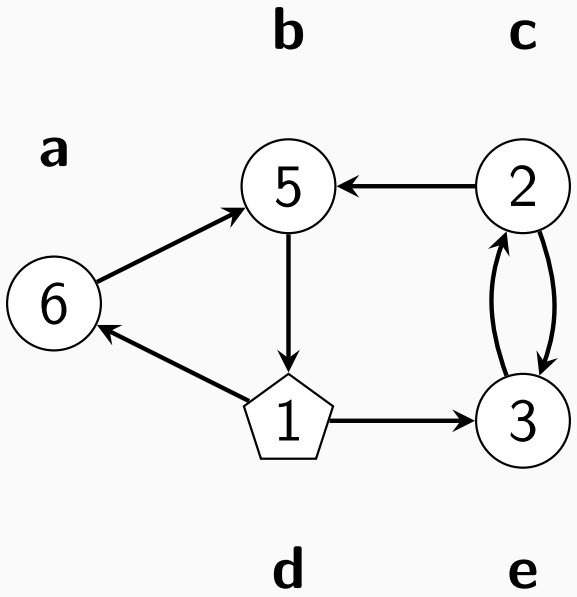
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



A player wins a vertex v if they have a positional strategy σ such that every play from v consistent with σ is won by that player.

PARITY GAMES

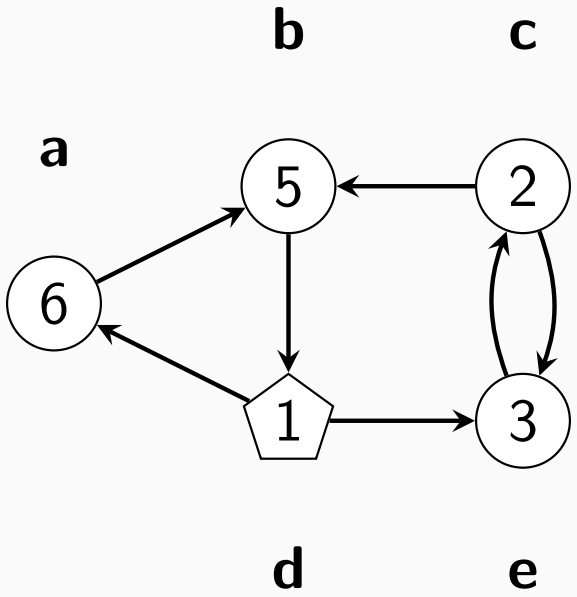
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



A player wins a vertex v if they have a positional strategy σ such that all cycles in the induced graph $\mathcal{G}[\sigma]$ are won by that player.

PARITY GAMES

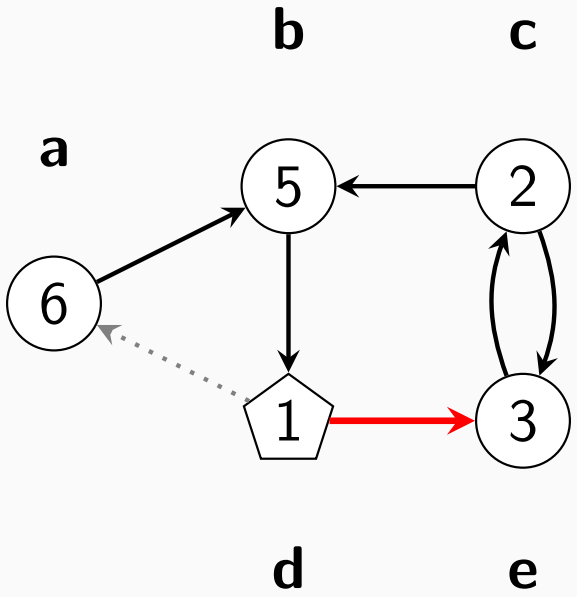
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



Which vertices are won by which player?

PARITY GAMES

- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



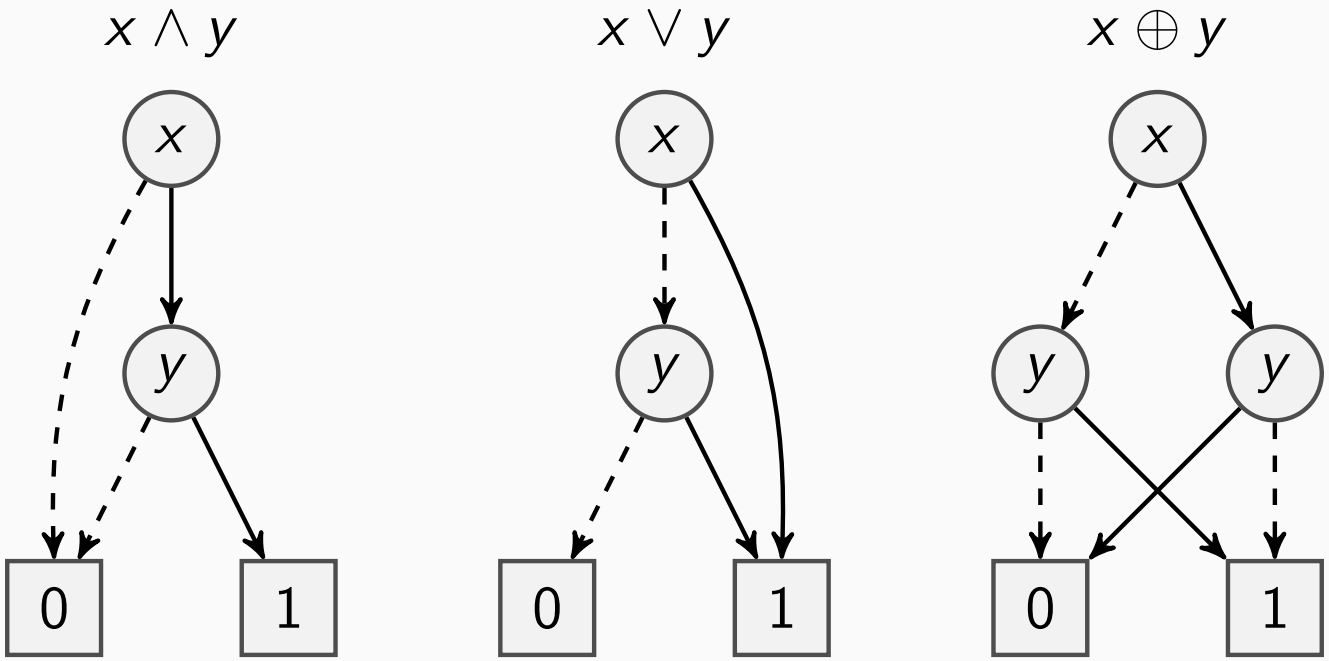
Player Odd wins with strategy $\{\, \mathbf{d} \to \mathbf{e} \,\}$
Only two (Odd) cycles remain

Solving a parity game

- Compute the winning regions $W_{\bigcirc}$ and $W_{\pentagon}$
  - For some applications: who wins the initial vertex/vertices?
- Compute the winning positional strategies $\sigma_{\bigcirc}$ and $\sigma_{\pentagon}$
  - how the controller can ensure the property
  - a counterexample that shows how the property cannot be satisfied

BINARY DECISION DIAGRAMS

- A BDD is a directed acyclic graph encoding a $\mathbb{B}^k \to \mathbb{B}$ function
- Paths represent valuations of $\mathbb{B}^k$
- Path to $\boxed{1}$, then valuation is in the set
- Typical operations are $\cup$, $\cap$, $\forall$, $\exists$, $\neg$



TOWARDS SYMBOLIC LTL SYNTHESIS

Motivation: LTL synthesis

- Incomplete transition system + Input/Output signals + LTL formula
- Output: a controller s.t. the whole system fulfills the specification

General procedure

- Translate LTL formula into a deterministic parity automaton
- Split the automaton on input and output transitions
  - Read input letter on transition to intermediate state
  - Read output letter on transition to actual successor
  - This is an alternating automaton, or, a parity game
- Solve the parity game, then use the positional strategy of the "controller player" to produce a controller guaranteed to fulfill the specification

SYMBOLIC ALGORITHMS

Motivation for symbolic methods

- State space explosion
- Size of the incomplete system, e.g., due to composition
- Two exponential steps from LTL to deterministic parity automaton
- Binary decision diagrams (BDDs) under the right circumstances concisely represents sets of states/vertices

SYMBOLIC ALGORITHMS

Motivation for symbolic methods

- State space explosion
- Size of the incomplete system, e.g., due to composition
- Two exponential steps from LTL to deterministic parity automaton
- Binary decision diagrams (BDDs) under the right circumstances concisely represents sets of states/vertices

The right circumstances

- Nice "symmetries" (e.g. due to composition)
- Example: dining philosophers, buffers, etc.
- Good variable ordering is *critical*

SYMBOLIC ALGORITHMS

How to approach algorithms with BDDs in mind?

- Consider algorithm as operating on sets of vertices or on functions that map vertices to values
- Avoid enumerating / processing individual vertices
- Core data structures are binary decision diagrams:
  - e.g. $s \in S \Leftrightarrow f(s) = 1$ using BDDs
  - e.g. $\mathrm{pr}(v) = p$ using ADDs/MTBDDs

SYMBOLIC PARITY GAME ALGORITHMS

Existing work

- Current literature implements with BDDs:
  - Zielonka's recursive algorithm
  - Small progress measures
  - Priority promotion
  - `BFL`
  - `APT`
- Only Kant et al produces the winning strategies for Zielonka's recursive algorithm, but not explicitly, and the focus of their work lies on generating the symbolic parity games from μ-calculus model checking

Work on "nested fixpoint" family

- μ-calculus nested fixpoint algorithms are explicitly set based
- They are extremely fast in the explicit form
- Until recently, these algorithms did not yield straight-forward strategies

FIXPOINT ITERATION ALGORITHMS

Four current algorithms

- (**APT**) Via weak alternating automata
  - Encoding to weak alternating automata: *Kupfermann, Vardi, 1998*
  - As parity game solver: *Di Stasio, Murano, Perelli, Vardi, 2016*
  - Does not give winning strategies!
- (**BFL**) Via μ-calculus translation of Zielonka's recursive algorithm
  - Encoding to μ-calculus: *Walukiewicz, 1996*
  - As parity game solver: *Bruse, Falk, Lange, 2014*
  - Has complicated method to compute winning strategies!
- (**DFI**) As the fixed point of distractions
  - *Van Dijk, Rubbens, 2019*
  - Uses frozen sets to fix the winning strategy
  - This also speeds up the computation
- (**FPJ**) Using a justification graph
  - *Lapauw, Bruynooghe, Denecker, 2020*
  - Uses a justification graph to store the winning strategy
  - This also speeds up the computation

FIXPOINT ITERATION ALGORITHMS

Outline

- All four algorithms compute the same set of distractions: even priority vertices that are won by player Odd and vice versa
- "Update whether vertices are a distraction by looking 1 step ahead"
- A vertex is a one-step distraction if:
  - it has even priority and is won by player Odd in 1 step
  - it has odd priority and is won by player Even in 1 step
- Record "distraction set" Z
- Initially $Z = \emptyset$ (that is: won by the priority player)
- Nested fixed points: monotonically update Z_0, then Z_1, etc.
- After adding vertices to Z_p, reset all $Z_{<p}$ to $\emptyset$ and recompute

FIXPOINT ITERATION ALGORITHMS

Outline

- All four algorithms compute the same set of distractions
- "Update whether vertices are a distraction by looking 1 step ahead"
- Nested fixed points: monotonically update Z_0, then Z_1, etc.
- After adding vertices to Z_p, reset all $Z_{<p}$ to $\emptyset$ and recompute

Variations

- **APT** is just the base nested fixpoint
- **BFL** adds a mechanism to compute strategies with a secondary algorithm
- **DFI** adds "freezing" mechanism
- **FPJ** adds "justification" mechanism

IMPLEMENTATION CHOICES

Choices

- Prototype in Python
- Use a simple Python bridge to CUDD
- Standard operations, no special variable ordering, no special operations
- Encoding of a symbolic parity game
  - Encode vertices to variables using a naive binary encoding
  - Store vertex ownership as a simple set V_{even} and V_{odd}
  - Store a set V_p for every priority p
  - Store a set E to store the edges
- Store the set of distractions Z as a single set
- Several additional sets for each frozen set (one for each priority), or the justification graph, and the winning strategy

EMPIRICAL EVALUATION

Evaluation

- Four symbolic algorithms:
  - Zielonka's recursive algorithm (by Sanchez et al)
  - DFI-ns (without strategy computation)
  - DFI (with strategy computation)
  - FPJ (with strategy computation)
- Benchmarks
  - Obtained from the SYNTCOMP competition
  - Converted to a parity game using Knor and Oink
  - Then translated to a symbolic parity game using a naive binary encoding

EMPIRICAL EVALUATION

Statistics of benchmark sets used for empirical evaluation

Set	#games	avg n	max n	prio's	avg δ	avg #bdd nodes
Lily	23	409	3047	3-8	1.47	1064
AMBA	8	2461	18635	3-4	1.43	1780
ltl2dba	58	921	31717	4-7	1.49	1876
Arbiters	15	2650	20928	4	1.65	6068
Detector	2	98	120	4	1.47	373
Buffer	2	14	17	4	1.30	72
Load balancer	11	1149	4712	3-8	1.65	2589

EMPIRICAL EVALUATION

Cumulative time in sec. average over 5 runs used to solve all games

Set	computes strategy		no strategy	
	`dfi`	`fpj`	`dfi-ns`	`zlk`
Lily	0.38	0.54	**0.19**	0.39
AMBA	0.39	0.70	**0.29**	0.63
ltl2dba	21.11	16.13	**14.63**	15.08
Arbiters	48.17	23.98	32.48	**17.54**
Detector	0.0066	0.0068	**0.0046**	0.0047
Buffer	0.00082	0.00058	**0.00052**	0.00057
Loadbalancer	0.50	0.91	**0.37**	0.62

EMPIRICAL EVALUATION

Average Peak Live BDD Nodes (peak nodes per algorithm run)

Set	computes strategy		no strategy	
	`dfi`	`fpj`	`dfi-ns`	`zlk`
Lily	41647	51747	**37431**	44595
AMBA	34299	36261	**28143**	34729
ltl2dba	212242	263743	**183830**	326327
Arbiters	172235	231625	**153008**	218382
Detector	1400	1393	**1211**	1427
Buffer	244	244	**222**	253
Load balancer	50411	66389	**44407**	60264

Working with BDDs

- Different ways to encode a function, e.g., integer label on vertex
  - As sets V_0, V_1, V_2 for every label 1, 2, 3
  - Binary encoding of the pairs (v, p), with $\mathrm{pr}(v) = p$ iff (v, p) is in the set (and only one p for every v)
  - As a MTBDD/ADD (non-binary leaf, i.e., leaves are numbers instead of just true/false)
- Variable order is critical and difficult
- Order of operations has huge impact
- Using tailored operations has huge impact
- These are all directions for follow-up work

CONCLUSION/DISCUSSION

- Two new symbolic algorithms based on the explicit nested fixpoint algorithms DFI and FPJ
- All algorithms are "fast in practice"
- About 2× runtime to also compute strategies
- Lots of optimizations possible
  - Fully rewrite in C/C++ instead of Python
  - Use Sylvan instead of CUDD
  - Use tailored BDD operations
  - Experiment with the variable ordering
- Translate LTL to a symbolic DPA
- We need better benchmarks for LTL synthesis