# Reproducing parity game algorithms

Tom van Dijk – t.vandijk@utwente.nl

6 April 2024

# OUTLINE

- Background: parity games
- State of the art
- Goals of Oink
- Reproducibility Results and Discussion

What is the problem of parity games and why are there so many algorithms?

# PARITY GAME SOLVING

## What are parity games?

Parity games are two-player games on a graph used in formal verification.

## Why study parity games?

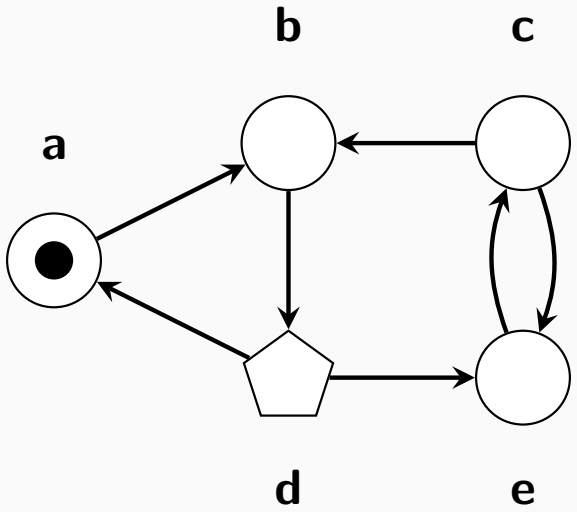
Solving a parity game is

- maybe in polynomial time (in **P**)
- since 2017 strictly below exponential time ("below" **NP**)
- important problem on the border between **P** and **NP**

... and faster algorithms are nice too ;)

# PARITY GAMES

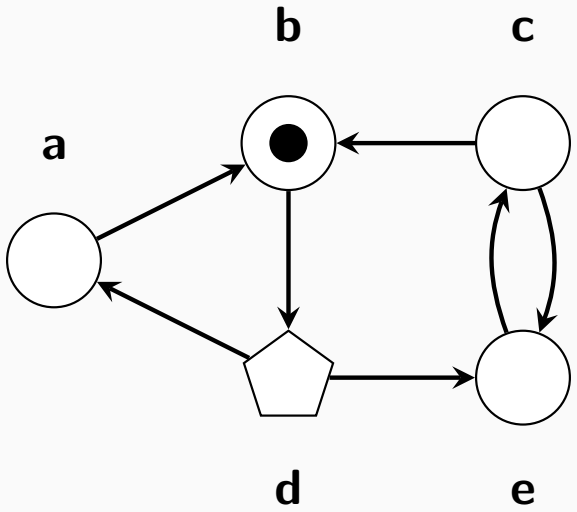
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a**

# PARITY GAMES

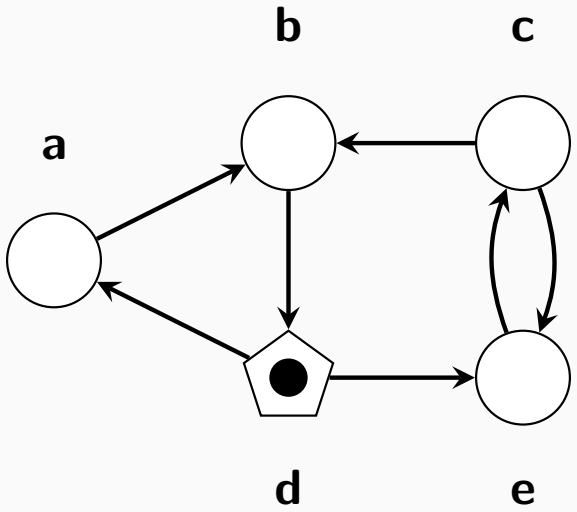
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b**

# PARITY GAMES

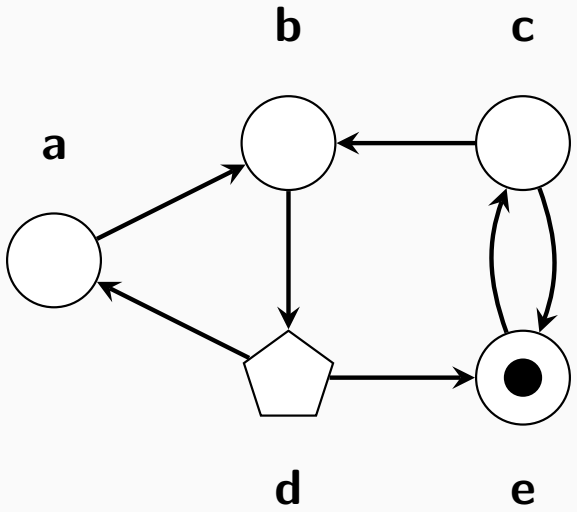
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d**

# PARITY GAMES

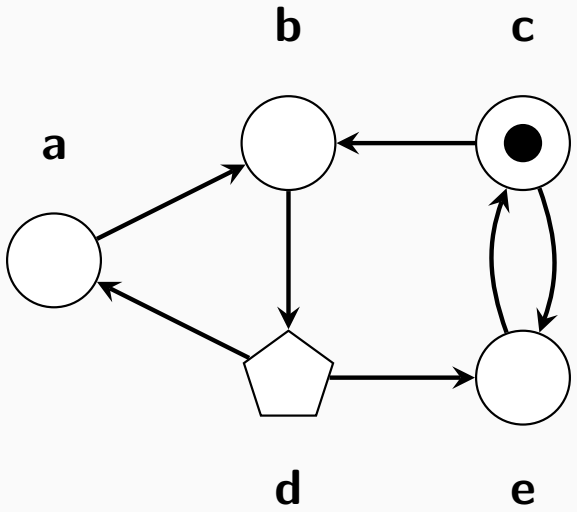
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e**

# PARITY GAMES

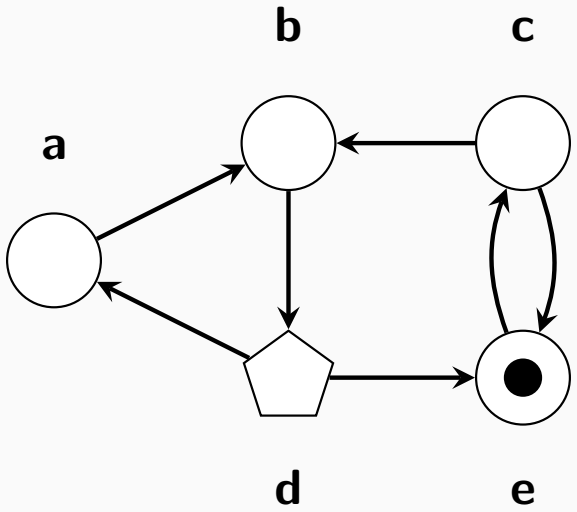
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c**

# PARITY GAMES

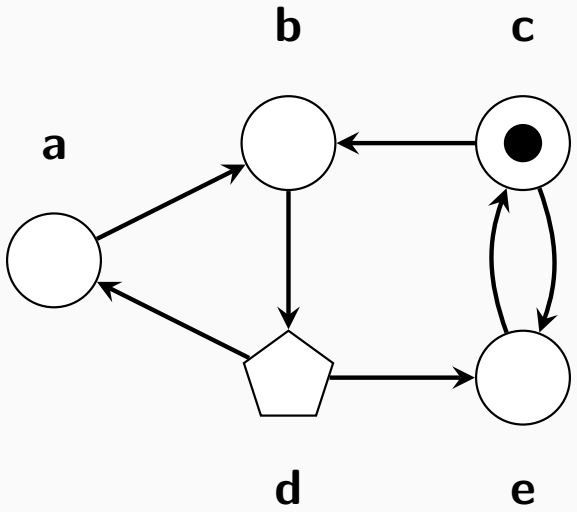
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e**

# PARITY GAMES

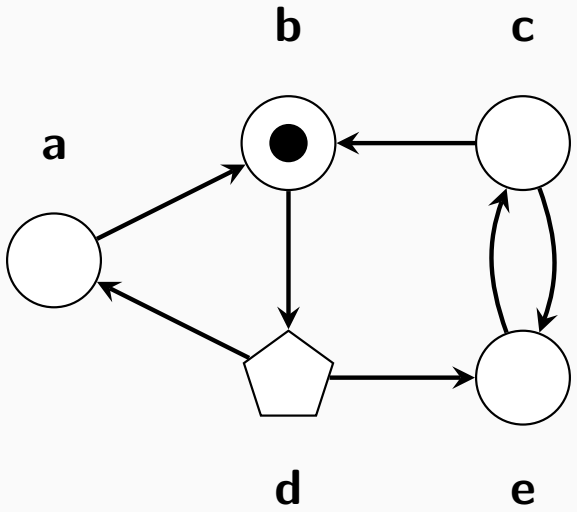
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e c**

# PARITY GAMES

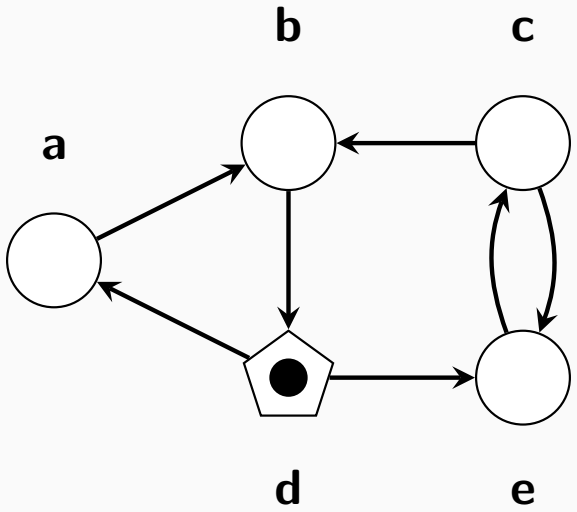
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e c b**

# PARITY GAMES

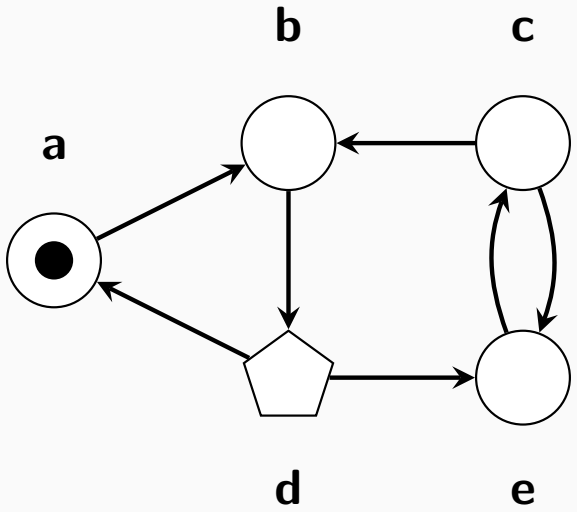
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e c b d**

# PARITY GAMES

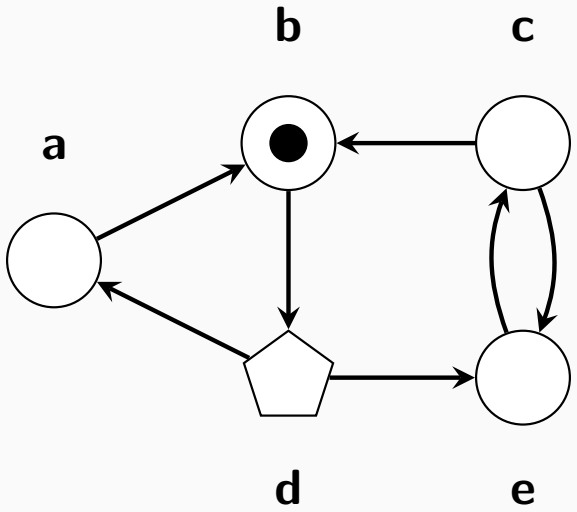
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e c b d a**

# PARITY GAMES

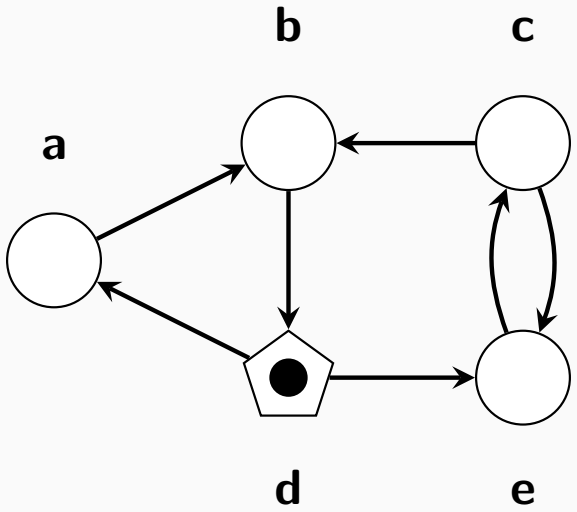
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e c b d a b**

# PARITY GAMES

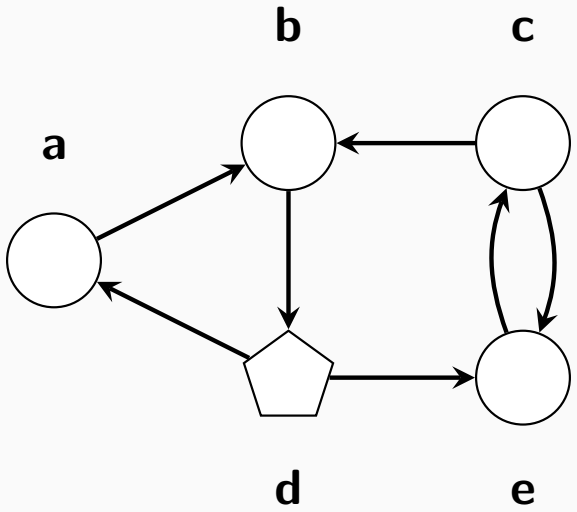
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




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

# PARITY GAMES

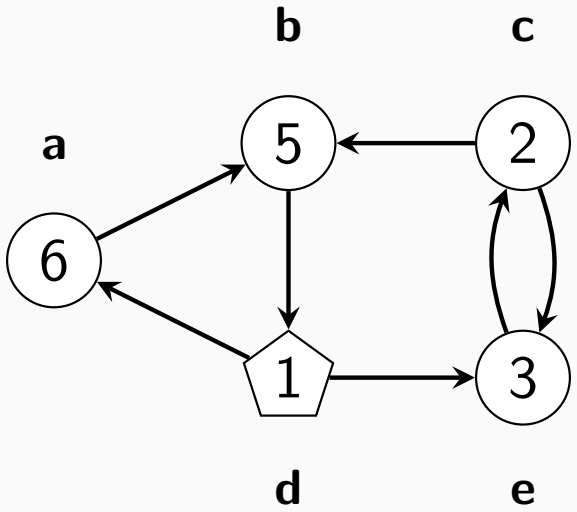
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




Who wins a play?

# PARITY GAMES

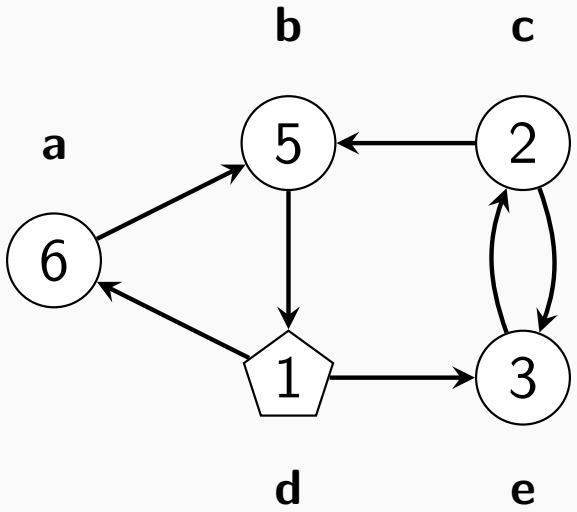
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




- 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

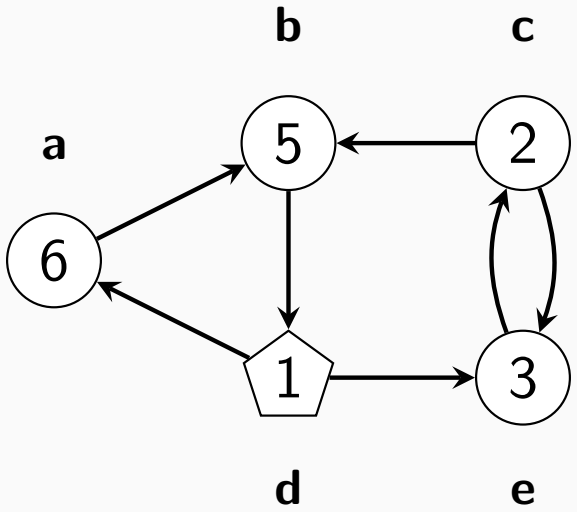



The play $\pi$: 6 5 1 3 2 3 2 5 $(1\ 6\ 5)^{\omega}$

Who wins this play?

# PARITY GAMES

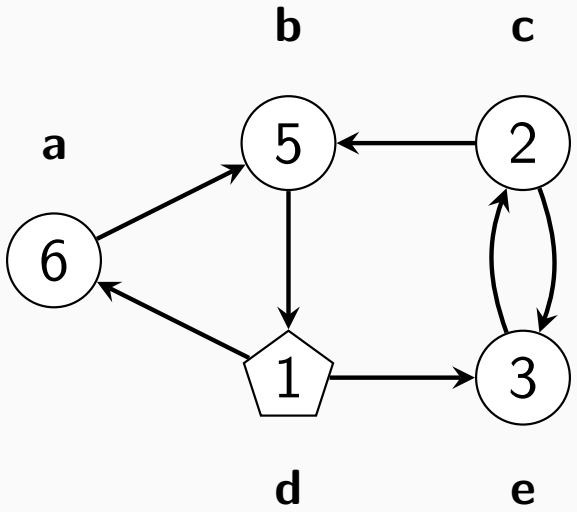
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




How do we determine who wins a vertex?

# PARITY GAMES

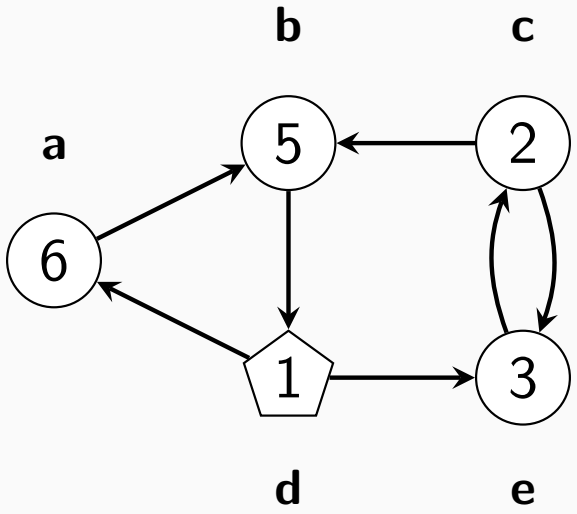
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




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

# PARITY GAMES

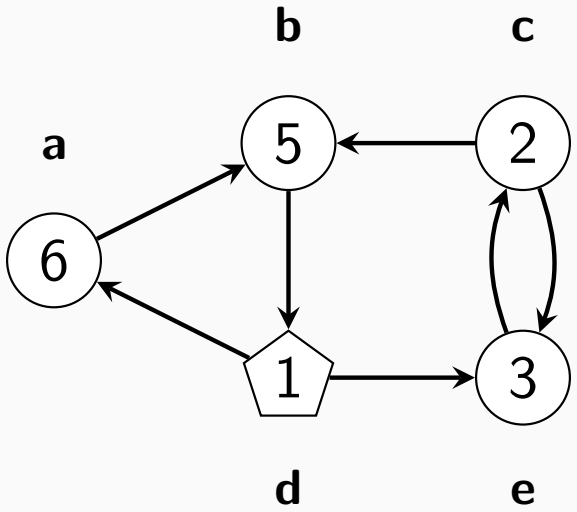
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




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

# PARITY GAMES

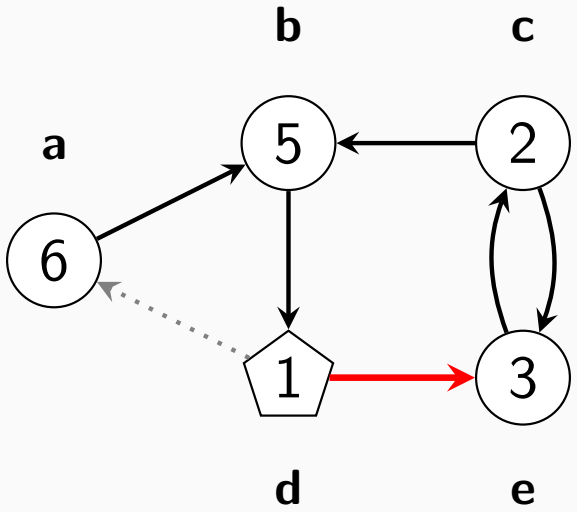
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




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

# PARITY GAMES

## Solving a parity game

- Compute the winning regions $W_{\bigcirc}$ and $W_{\pentagon}$
- Compute the winning positional strategies $\sigma_{\bigcirc}$ and $\sigma_{\pentagon}$

# PARITY GAME ALGORITHMS

## Decomposition/Attractor-based

*Properties of sets of vertices* computed with attractors.

- recursive algorithm (+ quasi-polynomial variation)
- priority promotion
- tangle learning

## Local value improvement

Improve the *value of individual vertices* based on direct successors.

- small progress measures
- succinct progress measures (quasi-polynomial)
- ordered progress measures (quasi-polynomial)
- strategy improvement
- nested fixed point iteration

## Decomposition-based algorithms

- Use the attractor computation to decompose the game into regions
- Refine/improve decomposition until dominion found

- First attract for some player $\alpha$ to the "best vertex", then continue decomposition of the remaining game

- Region invariant
  - Player $\alpha$ wins all plays that stay in the region
  - Player $\overline{\alpha}$ either stays in the region (and loses), or visits the target vertex, or escapes to a higher region of player $\alpha$

# PARITY GAME ALGORITHMS

## Zielonka's recursive algorithm (1998)

Bottom-up attract to lower opponent regions in a strict order.
After each change, recompute lower regions of the loser.

## Priority promotion (2016)

Merge locally closed regions upwards.
Attract to the merged region, then recompute lower regions.

## Tangle learning (2018)

The attractor now also attracts using tangles.
Every iteration finds new tangles.
Use new tangles to that refine the decomposition.

# PARITY GAME ALGORITHMS

## Local value improvement

- Every vertex has some value (e.g. "how good for Even")
- Locally improve each vertex based on the value of the successors
- Value "propagates" through the game
- Intuition: playing a game in reverse
- Challenge: reasoning about cycles and nested cyclic structures

# PARITY GAME ALGORITHMS

## Progress measures (1998)

Players play their "best" game backwards, each vertex having a monotonically increasing value representing the current play value.

## Strategy improvement (2000)

Players take turns improving their strategy. That is:

- Players start with random strategies
- Opponent maximally improves their strategy (monotonically decreasing)
- Main player improves their strategy once, then repeat
- Best response always monotonically increases value
- Strategies are improved based on best successor (local property)

# PARITY GAME ALGORITHMS

- 1998: Zielonka's recursive algorithm, small progress measures
- 2000: strategy improvement
- many different variations of strategy improvement
- parallel versions of spm and si
- 2016-: new decomposition based: priority promotion, tangle learning
- 2017-: quasipolynomial algorithms
- **2018: Oink**

Various implementations in **PGsolver**, in the OCaml language.

# OINK

## Goals

- Compare the current state of the art, with modern C implementations of the representative algorithms
- Exact reproducibility was not the priority, the goal was reimplementation, possibly with improvements
- Understand the algorithms/approaches better, understand the mechanics behind the algorithms, hoping to combine approaches
- Eventually come up with better algorithms, either for practical performance or better theoretical complexity

# FINDING IMPLEMENTATIONS

- Sometimes easy: PGsolver contains a lot of implementations (as reference)
- Some implementations on Github. Sometimes requires a bit of detective work, not/barely referenced in papers.
- One tool was a part of the mCRL2 toolset. (you need to know where to look)
- Some implementations could not compile (properietary framework)
- Some implementations had unexpected performance problems
- Some websites are now offline.

# BENCHMARKS

**Very major challenge**:

- many papers use different methods to compare
- often benchmarks are a **small sample** of random games – hard to repeat, actual files unavailable, sensitive to bias/luck
- do we even want to use random games?

# BENCHMARKS

**Very major challenge**:

- many papers use different methods to compare
- often benchmarks are a **small sample** of random games – hard to repeat, actual files unavailable, sensitive to bias/luck
- do we even want to use random games?
- we use the Keiren benchmarks (2015), randomly generated games, and reactive synthesis benchmarks (more recent)

# COMPARISONS

- Algorithms are often compared to the PGsolver implementation of Zielonka's recursive algorithm
- Several papers already demonstrated major improvements
- Our study shows up to 1000$\times$ faster with optimized C version
- Worse: cross-language comparisons

# REPLICATING PARALLEL SI

**Success story**:

- 2017 paper with parallelized strategy improvement (Fearnley)
- impressive performance (esp. GPU)
- my sequential version roughly same performance
- my parallel version was faster (using different technique)
- unfortunately: didn't compare/replicate GPU version
- (note: parallel programming less important in practice)

# PROGRESS MEASURES

- Quasi-polynomial algorithms are difficult to implement
- OPM has vague implementation details that turn out incorrect too!
- SuccPM had no implementation whatsoever (except on Github, by someone else)
- Small variations (iteration/queue/stack) have big effects

# RECURSIVE ALGORITHM

- Many different available Zielonka implementations
- Version inside mCRL2 already state-of-the-art (vs PGsolver)
- Java version of SPGSolver did not run well in replication

# OTHER ALGORITHMS

- Priority promotion: quite a bit faster than ZLK in PGsolver, less so in Oink
- APT: much faster in PGsolver (compared to ZLK) but much less so in Oink
- Influence of data structures and memory locality.

# TAKEAWAYS (PART 1)

- Often implementations are not available or not available anymore.
- Empirical evaluation often lacks inputs, logs / raw data, actual implementations
- Implementation “details” have major impact
- Programming language can matter; compiler version can matter

# TAKEAWAYS (PART 2)

- Artifact making can be tricky for novices
- Shepherding important, especially when artifacts are optional
- Easier if reproducibility (one-button scripts) are a goal from the start

# TAKEAWAYS (PART 3)

- Having a common benchmark set is very useful
- Randomly generated benchmarks are terrible
- Artificially generated "hard" cases are terrible
- **Also**: “hard” games can depend on minor details in the implementation!

# TAKEAWAYS (PART 4)

- Licenses???

*This slide intentionally left empty.*