Solving Parity Games with Oink

Tom van Dijk (JKU Linz)

TACAS, Thessaloniki, 17 April 2018

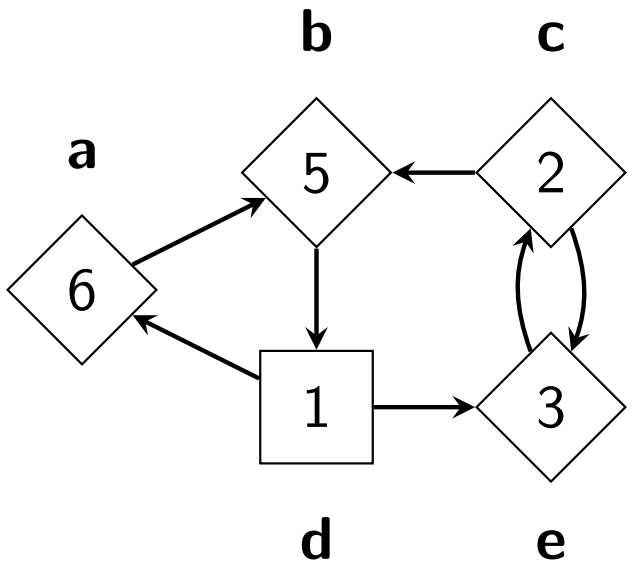
TL; DR

- Parity games
  Solve model checking and synthesis by playing a game.
  They are simple and quite addictive.
- Goal 1: Solving algorithms
  Many algorithms exist, roughly either based on attractor computation or on local value iteration.
- Goal 2: The tool Oink
  Modern & fast implementation of parity games solvers.
  Easy to use, easy to extend.
- Goal 3: Compare algorithms
  The attractor-based solvers are fastest in practice.

Parity Games

Definition

- A parity game is a directed graph
- Each vertex has a priority $\{0, 1, 2, \ldots, d\}$
- Each vertex is owned by player Even $\diamond$ or player Odd $\square$



Parity Games

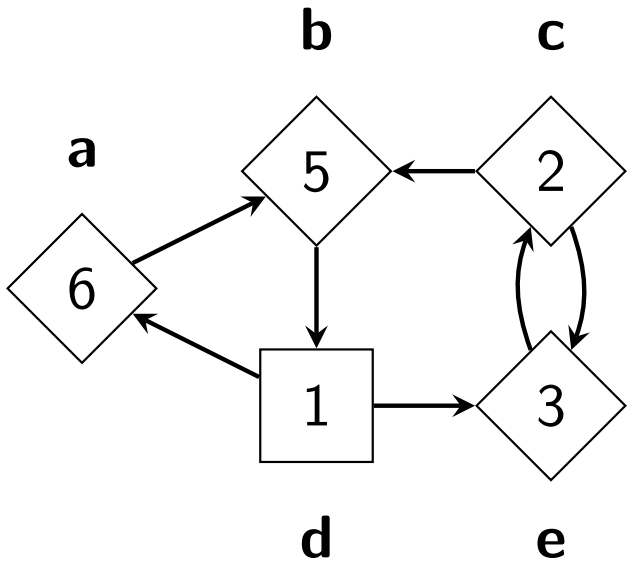
Definition

- A parity game is a directed graph
- Each vertex has a priority $\{0, 1, 2, \ldots, d\}$
- Each vertex is owned by player Even $\diamond$ or player Odd $\square$

Rules of the game

- They play an infinite game along the graph.
- Owner of each vertex determines the next move.
- Winning condition:
  The parity of the highest priority seen infinitely often.
- Player Even wins if this priority is even.
- Player Odd wins if this priority is odd.

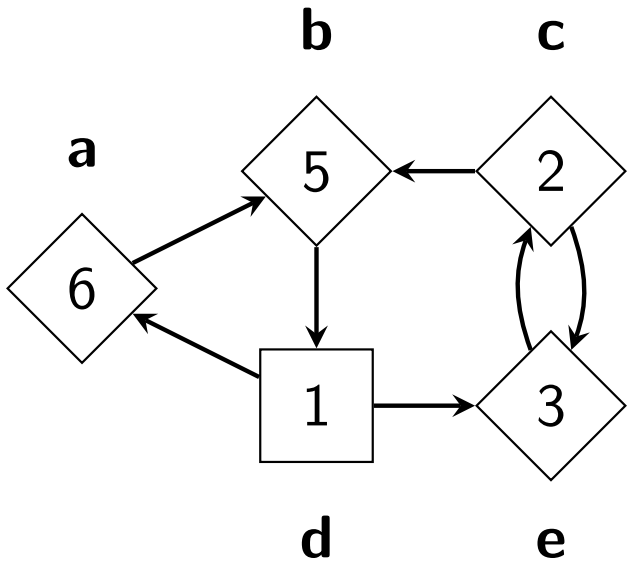
Parity Games



Example parity game

- Even $\diamond$ controls **a**, **b**, **c**, **e**. Odd $\square$ controls **d**.
- They play an infinite game along the edges of the game.
- Who wins each vertex?

Parity Games



Example parity game

- Even $\diamond$ controls **a**, **b**, **c**, **e**. Odd $\square$ controls **d**.
- They play an infinite game along the edges of the game.
- Player Odd wins from all vertices with strategy $\{\mathbf{d} \to \mathbf{e}\}$.

Parity Games

Why are parity games practically relevant?

- Quite expressive: parity games capture the expressive power of nested least and greatest fixpoint operators.
- Polynomial-time equivalent to:
  - modal μ-calculus model-checking (CTL, LTL...)
  - checking emptiness of non-deterministic parity tree automata
  - solving boolean equation systems
- Backend for synthesis, e.g. LTL synthesis.
- Also equivalence checking, propositional proof complexity, etc.

Parity Games

Why are parity games theoretically interesting and addictive?

- There is probably a (polynomial) solution in P!

Why?

- The problem is in **NP** $\cap$ **co-NP**.
  - If NP-complete, then **NP** $=$ **co-NP**.
  - Therefore very unlikely to be NP-complete.
- The problem is even in **UP** $\cap$ **co-UP**.
  - A (weak) subclass of **NP** $\cap$ **co-NP**; also integer factorization.

Parity Games

(Incomplete list of) published algorithms

McNaughton/Zielonka	$O(e \cdot n^d)$ and $O(2^n)$	1998
Small Progress Measures	$O(d \cdot e \cdot (n/d)^{d/2})$	1998
Strategy Improvement	$O(n \cdot e \cdot 2^e)$	2000
Dominion Decomposition	$O(n^{\sqrt{n}})$	2006
Big Step	$O(e \cdot n^{d/3})$	2007
APT	$O(n^d)$	2016
Priority Promotion	Exponential	2016
Quasi-Polynomial	$O(n^{6+\log d})$	2016
Tangle Learning (at CAV)	?	2018

Outline

Parity Game Algorithms

Descriptions in the paper

- Aim of contribution: describe 5 algorithms intuitively.
- Improved operational description of *small progress measures*
- Complementing description of *priority promotion*

Roughly two types

- Local value iteration
  Based on locally improving the value of individual vertices by looking at their successors.
- Attractor-based
  Based on properties over sets of vertices computed with attractors.

Parity Game Algorithms

Local value iteration

- Every vertex has some value.
- Locally improve each vertex based on the successors.

Example: Strategy Improvement (2000)

Players take turns improving their strategy until a fixed point.

Example: Progress Measures (1998, 2016, 2017)

Players play the game backwards w.r.t. how good the optimal game so far is for one of the players.

Parity Game Algorithms

Attraction-based algorithms

- Partition the game into regions using attractors.
- Start with the highest priority (top-down).
- Each region is tentatively won by one player.
- Refine winning regions until dominion found.

Example: Zielonka's Recursive Algorithm (1998)

Attract higher regions downward after computing lower regions.
If your opponent attracts from your region, recompute your part.

Example: Priority Promotion (2016)

Merge regions upwards when the region is closed (in the subgame).
Then recompute lower regions.

Outline

Oink

- Modern implementation of parity game algorithms
  - Zielonka's Algorithm (with optimizations; parallel)
  - Small progress measures (with optimizations)
  - Priority Promotion (different versions)
  - Strategy Improvement (parallel)
  - QPT progress measures
- The usual preprocessing algorithms
  - Inflation and compression
  - Remove self-loops
  - Detect winner-controlled winning cycles
  - SCC decomposition
- `https://www.github.com/trolando/oink`
- Simple to use/extend library in C++

Oink

Easy to use

```
#include "oink.hpp"

pg::Game parity_game;
parity_game.parse_pgsolver(cin);

pg::Oink solver(parity_game);
solver.setSolver("zlk");
solver.run();

parity_game.write_sol(cout);
```

Easy to extend

- Implement Solver interface
- Add one line to `solvers.cpp`

Empirical evaluation

Benchmarks

- Benchmarks by Jeroen Keiren.
- 313 model checking and 216 equivalence checking games.
- Furthermore 420 random games of up to 20000 vertices.

Solvers

- Oink *
- PGSolver (Friedmann et al)
- pbespgsolve (Willemse et al)
- parallel-si (Fearnley) *
- SPGSolver (di Stasio, Arcucci et al) *
- * = Both sequential and multi-core versions

Empirical evaluation

	Model checking		Equiv checking		Random games		Total	
zlk-1	113	0	560	0	**5**	0	**679**	0
zlk-8	111	0	781	0	26	0	918	0
pbeszlk	**67**	0	**533**	0	340	0	941	0
zlk	87	0	903	0	**5**	0	995	0
spg-seq	1262	1	2639	2	728	0	4629	3
spg-mc	474	0	4304	2	51885	39	56663	41
pgzlk	42881	32	44469	35	36859	19	124210	86

Runtimes in sec. (PAR2) and number of timeouts (10 minutes) of the four solvers PGSolver (`pgzlk`), SPGSolver (`spg`), pbespgsolve (`pbeszlk`) and Oink (`zlk`).

Empirical evaluation

	Model checking		Equiv checking		Random games		Total	
psi-8	956	0	**1474**	0	**551**	0	**2981**	0
psi	**889**	0	4085	1	565	0	5540	1
psi-1	1771	1	6640	4	613	0	9024	5
parsi-seq	1897	1	6596	4	1697	0	10190	5
parsi-mc8	1939	1	4885	2	47217	39	54041	42
parsi-mc1	3908	2	10508	7	47633	39	62049	48
pgsi	101137	78	69492	55	39020	27	209649	160

Runtimes in sec. (PAR2) and number of timeouts (10 minutes) of the three solver PGSolver (`pgsi`), the solver by Fearnley with sequential (`parsi-seq`) and multi-core variants, and Oink with sequential (`psi`) and multi-core variants.

Empirical evaluation

	Model checking		Equiv checking		Random games		Total	
spm	**3328**	1	**9668**	3	112556	93	**125552**	97
qpt	86000	67	50617	36	**45926**	36	182542	139
pbesspm	27375	21	36510	27	123200	101	187085	149
pgspm	67832	51	40185	30	214409	169	322426	250

Runtimes in sec. (PAR2) and number of timeouts (10 minutes) of PGSolver (`pgspm`), pbespgsolve (`pbesspm`) and `spm` and `qpt` in Oink.

Empirical evaluation

	Model checking		Equiv checking		Random games		Total	
pp	**149**	0	**543**	0	**10**	0	**702**	0
ppp	158	0	592	0	**10**	0	759	0
rr	185	0	596	0	**10**	0	791	0
dp	214	0	596	0	12	0	822	0
rrdp	197	0	644	0	11	0	852	0

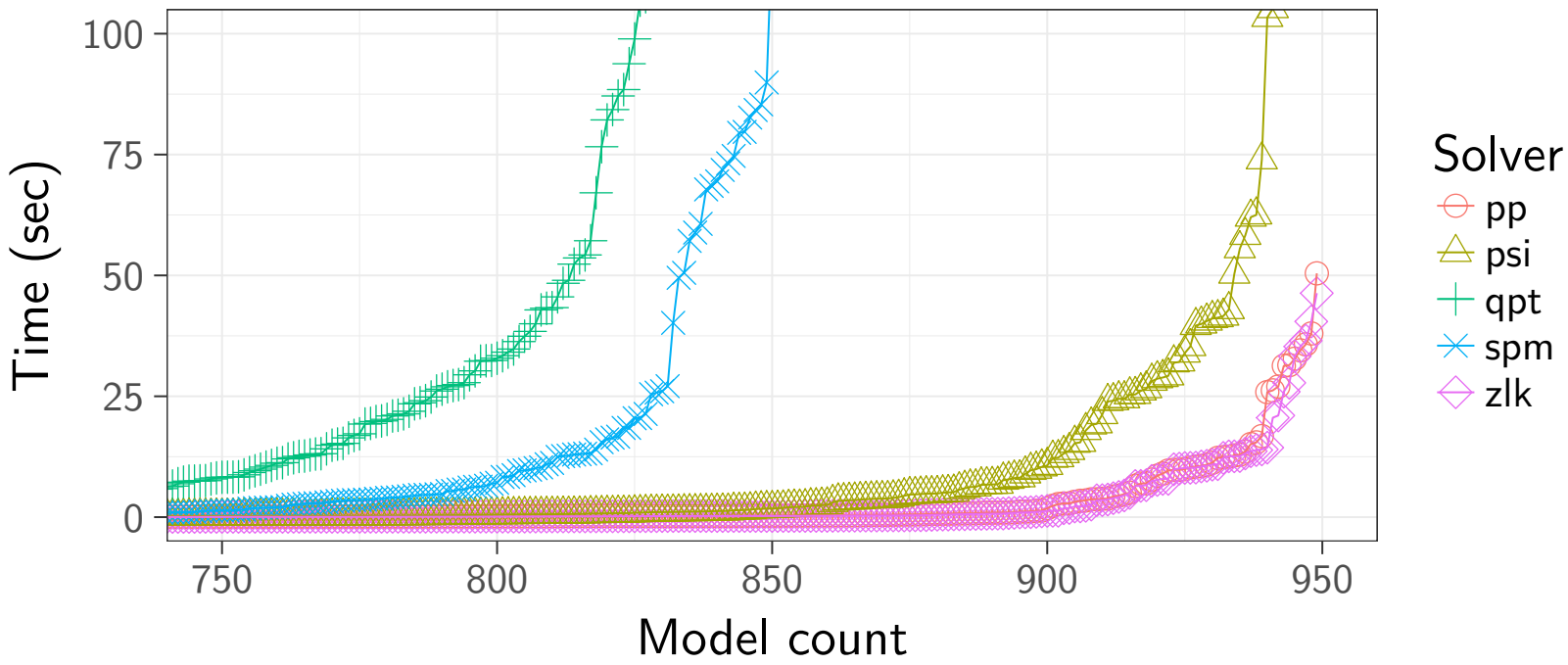
Runtimes in sec. (PAR2) and number of timeouts (10 minutes) of the five priority promotion solvers in Oink.

Empirical evaluation

	Model checking		Equiv checking		Random games		Total	
zlk	152	0	**474**	0	**8**	0	**634**	0
pp	**149**	0	543	0	10	0	702	0
psi	308	0	2559	0	830	0	3697	0
spm	1228	0	3648	0	104689	87	109565	87
qpt	42284	33	44796	34	42480	33	129560	100

Runtimes in sec. (PAR2) and number of timeouts (10 minutes) of the five sequential solvers in Oink

Empirical evaluation



A cactus plot of five sequential solvers implemented in Oink. The plot shows how many games are (individually) solved within a certain amount of time.

Conclusions

- Parity games are relevant and addictive.
- Fast moving field with great progress in the last few years.
- Oink is a competitive implementation.
- Attractor-based algorithms perform best.
- `https://www.github.com/trolando/oink`

	Model checking		Equiv checking		Random games		Total	
zlk	152	0	**474**	0	**8**	0	**634**	0
pp	**149**	0	543	0	10	0	702	0
psi	308	0	2559	0	830	0	3697	0
spm	1228	0	3648	0	104689	87	109565	87
qpt	42284	33	44796	34	42480	33	129560	100