Fixed-points, Tangles, Distractions in Parity Games

Tom van Dijk, Bob Rubbens

GandALF 2019

Overview

- *Two papers...*
  - Fixed-point algorithms for solving parity games
  - An exponential counterexample to attractor-based solvers
- *One talk...*
  - Goal: understanding parity game algorithms
  - Tangles in parity game algorithms
  - Distractions in parity game algorithms

Overview

Fixed-point algorithms for solving parity games

- State of the art:
  - (`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!
  - (`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!
- Contributions
  - We show that `BFL` and `APT` are equivalent
  - We propose a novel interpretation based on distractions
  - We find easy strategy derivation
  - Simple to implement and fastest for model-checking parity games

Overview

A Parity Game Tale of Two Counters

- Context
  - Broad goal: develop polynomial parity games solution
  - Using concepts like tangles, distractions
  - Many types of algorithms that deal with distractions differently
  - If not polynomial, deepen understanding with *strong* counterexamples
- Contributions
  - Parameterized parity game family for:
    - Recursive algorithm (Zielonka)
    - Priority promotion & variations
    - Tangle learning

    All attractor-based algorithms that recognize a distraction by finding that the opponent can attract the distraction
  - Based on two interleaved binary counters of N bits

Parity Games

- Context of parity games: formal verification of systems
  - Verify if a system implements the specification
  - Synthesize a controller that follows the specification
- Verification and synthesis as a game
  - player 0 wants to prove (or synthesize)
  - player 1 wants to refute
  - players make *choices* (player 0: $\exists, \vee, \bigcirc$, player 1: $\forall, \wedge, \Box$)
- Interesting systems often "run forever" (reactive systems)
  - when a car arrives, eventually the traffic light turns green
  - the reset button always works
  - "X is true until Y is true"
  - "X never happens before Y"

Hence: properties about infinite runs of finite-state systems

Parity Games

Why do we want to solve parity games?

- Expressive power of nested least and greatest fixpoint operators
- Linear equivalence with modal μ-calculus model-checking
  (CTL, LTL, CTL*, ...)
- Backend for LTL model checking and LTL synthesis
  Important industrial applications (PSL, SVA)

Parity Games

Why do we want to solve parity games?

- Expressive power of nested least and greatest fixpoint operators
- Linear equivalence with modal μ-calculus model-checking
  (CTL, LTL, CTL*, ...)
- Backend for LTL model checking and LTL synthesis
  Important industrial applications (PSL, SVA)

Open question: Is solving parity games in **P**?

- It is in $\mathbf{NP} \cap \mathbf{co\text{-}NP}$ and in $\mathbf{UP} \cap \mathbf{co\text{-}UP}$
- Hot topic! Recently new quasi-polynomial algorithms and even quasi-polynomial lower bounds for many algorithms!

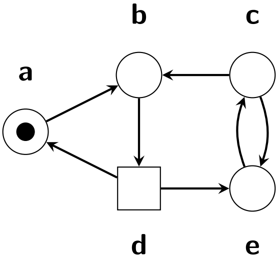
Parity Games

(Incomplete list of) algorithms

McNaughton/Zielonka	$\mathcal{O}(e \cdot n^d)$, $\mathcal{O}(2^n)$	1998
Small Progress Measures	$\mathcal{O}(d \cdot e \cdot (n/d)^{d/2})$	1998
Strategy Improvement	$\mathcal{O}(n \cdot e \cdot 2^e)$	2000
Dominion Decomposition	$\mathcal{O}(n^{\sqrt{n}})$	2006
Big Step	$\mathcal{O}(e \cdot n^{d/3})$	2007
Fixed Point: APT, BFL, DFI	$\mathcal{O}(n^d)$	2014 – 2019
Priority Promotion	$\Omega(2^{\sqrt{n}})$	2016
Quasi-Polynomial (several)	$\mathcal{O}(n^{6+\log d})$	2016 – 2018
Tangle Learning	$\Omega(2^{\sqrt{n}})$	2018
Recursive Tangle Learning	$\Omega(2^{\sqrt{n}})$	ongoing
Progressive Tangle Learning	$\Omega(2^{\sqrt{n}})$ (!)	ongoing

Parity Games

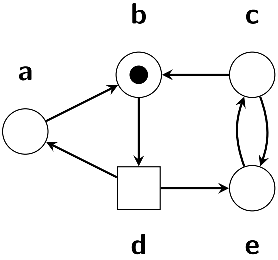
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



The play π: **a**

Parity Games

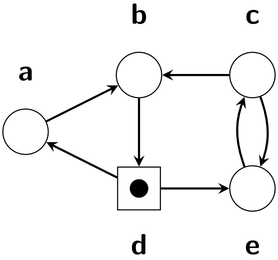
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



The play π: **a b**

Parity Games

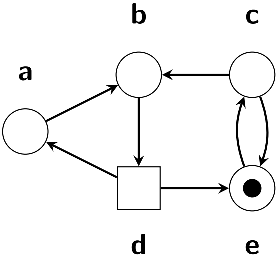
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



The play π: **a b d**

Parity Games

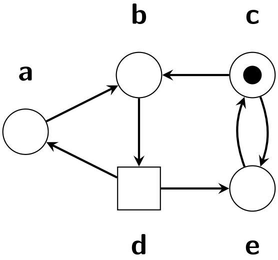
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



The play π: **a b d e**

Parity Games

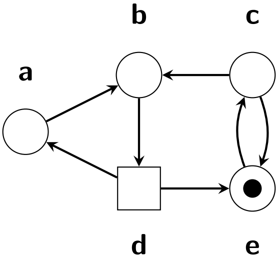
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



The play π: **a b d e c**

Parity Games

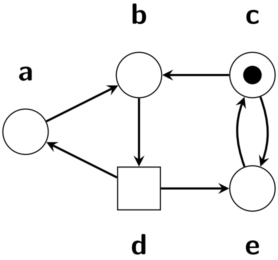
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



The play π: **a b d e c e**

Parity Games

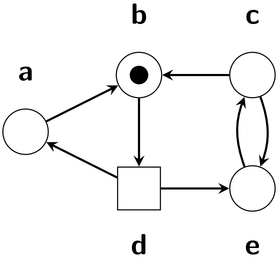
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



The play π: **a b d e c e c**

Parity Games

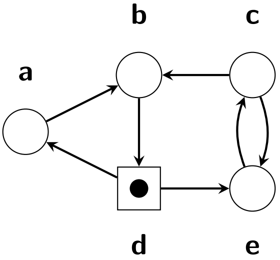
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



The play π: **a b d e c e c b**

Parity Games

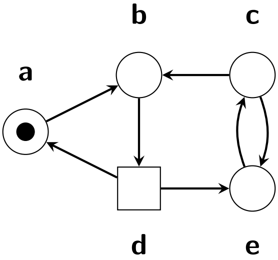
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



The play π: **a b d e c e c b d**

Parity Games

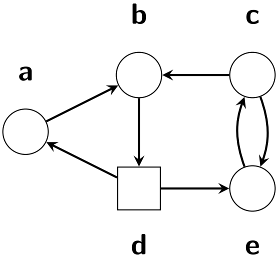
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



The play π: **a b d e c e c b d a**

Parity Games

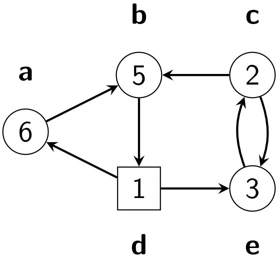
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



How do we determine who wins a play?

Parity Games

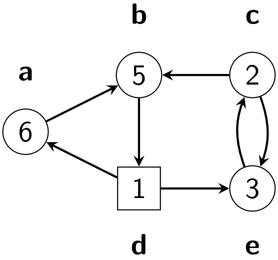
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



- Each vertex has a priority $\{0, 1, 2, \ldots, d\}$
- Highest priority seen infinitely often determines winner
- Player Even wins if this number is even

Parity Games

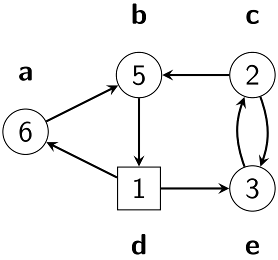
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



How do we determine who wins a vertex?

Parity Games

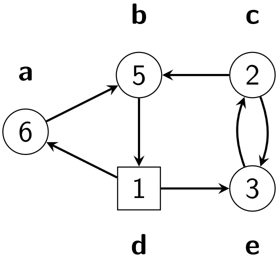
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



A player wins a (sub)game if they have a strategy to win all plays in the (sub)game, that is, such that the player wins all cycles that agree with the strategy.

Parity Games

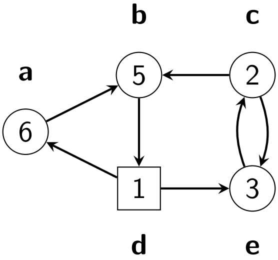
- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



Which vertices are won by which player?

Parity Games

- A parity game is played on a directed graph
- Two players: Even ○ and Odd □
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor



Player Odd wins all vertices with strategy $\{\mathbf{d} \rightarrow \mathbf{e}\}$

Parity Games

Basic facts of parity games

- Partition into $W_{\bigcirc}$ (won by Even) and $W_{\Box}$ (won by Odd)
- The winner α has a positional strategy $\sigma_\alpha \colon V_\alpha \to V$
- All plays consistent in W_α with σ_α are won by α
  That is: player α wins all cycles in $W_\alpha[\sigma_\alpha]$

Solving a parity game

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

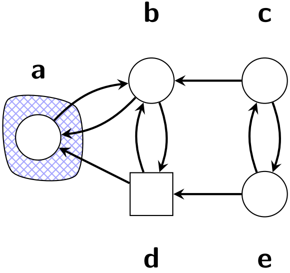
Attractor computation

- “Reachability but in a game”
- Compute all vertices from which player $\alpha \in \{\circ, \square\}$ can ensure arrival in a given target set
- Start with the target set Z, then iteratively add:
  - all vertices of α that can play to Z
  - all vertices of $\overline{\alpha}$ that can only play to Z

Parity Games

Example of attractor computation

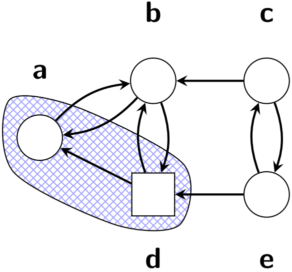
Computing the □-attractor to **a**



Initial set: $\{\,\mathbf{a}\,\}$
Can attract: **d** but not **b**

Example of attractor computation

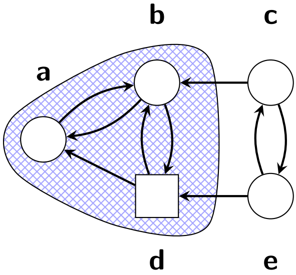
Computing the □-attractor to **a**



Current set: $\{\mathbf{a}, \mathbf{d}\}$
Can attract: **b** but not **e**

Example of attractor computation

Computing the □-attractor to **a**



Current set: $\{\mathbf{a}, \mathbf{b}, \mathbf{d}\}$
Can attract: neither **c** nor **e**

Parity Game Algorithms

Roughly two types

- Attractor-based
  Based on properties of *sets of vertices* computed with attractors.
  - McNaughton/Zielonka's recursive algorithm
  - Priority promotion
  - Tangle learning
- Local value improvement
  Based on locally improving/updating the *value of individual vertices* by looking at their successors.
  - Monotonic values:
    - small progress measures
    - succinct progress measures (quasi-polynomial)
    - ordered progress measures (quasi-polynomial)
  - Semi/non-monotonic values:
    - strategy iteration and many variations
    - nested fixed points iteration

Parity Game Algorithms

Attractor-based algorithms

- Partition the game into regions using attractors.
- Start with the highest priority (top-down).
- Each region is tentatively won by one player.
  - All plays that *stay* in the region are won by that player, and the opponent must leave the region either to a higher region of the same player, or via the “top” vertex of the region.
- Refine winning regions until dominion found.

Parity Game Algorithms

Zielonka's recursive algorithm (1998)

Attract, in a strict order, from higher regions to lower opponent regions.
After each change, recompute lower regions of the loser.

Priority promotion (2016)

Merge regions upwards when the region is closed (in the subgame).
Attract to the merged region, then recompute lower regions.

Tangle learning (2018)

The attractor now also attracts using tangles.
Every iteration of tangle learning finds new tangles that refine the partition.
Standard implementation: every closed region contains new tangles.

Parity Game Algorithms

Local value improvement

- Every vertex has some value (from the perspective of one player)
- *Intuition*: how good is the best game we currently know from there
- Important feature: value increases/decreases along paths *backwards*
- Locally improve each vertex based on the successors.
- *Intuition*: playing the game backwards

Parity Game Algorithms

Strategy iteration (2000)

Players take turns improving their strategy until a fixed point. One player maximally improves the strategy. Then the other player improves once, ensuring monotonic progress after each time the first player maximally improves their strategy.

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. The difference between the progress measures variations is in how they measure value.

Definition

A tangle is

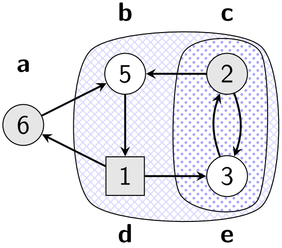
- a pair $T = (U, \sigma)$ where
  - $U \subseteq V$ is a nonempty set of vertices
  - $\sigma\colon U_\alpha \to U$ is a strategy for player $\alpha := \mathsf{pr}(U) \bmod 2$

  such that
- the induced subgame $\Game[U, \sigma]$ is strongly connected
- player α wins all cycles in the induced subgame $\Game[U, \sigma]$

Tangles

Tangle

A tangle is a strongly connected subgame for which one player has a strategy to win all cycles in the subgame.



A 5-dominion with a 5-tangle and a 3-tangle

Tangles

Tangle

A tangle is a strongly connected subgame for which one player has a strategy to win all cycles in the subgame.

Properties

- Player $\overline{\alpha}$ **must** escape (or lose inside the tangle)
- Player $\overline{\alpha}$ can reach all vertices of the tangle
  (and can thus choose which escape to take)
- Can view a tangle as "you must choose one of the escapes"
- Tangles can be nested (subtangles when player $\overline{\alpha}$ can avoid vertices)

Tangles vs dominions (winning regions)

- A closed tangle (no escapes) is a dominion
- Every dominion decomposes into a hierarchy of tangles

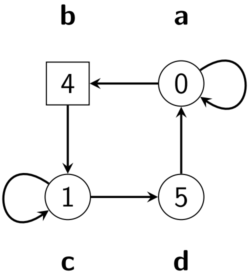
Tangles

Role of tangles

- Tangles affect every algorithm
  1. The opponent avoids some high value region
  2. Then discovers that the low value region is terrible
  3. Must choose one of the escapes
  4. Chooses the lowest (least bad) escape
- Every algorithm *MUST* reason about how the winner forces the loser to play to certain high value vertices, or, how the loser has no choice but to play to certain high value vertices: tangles!
- Recursive algorithm: the lower regions won by the opponent
- Priority promotion: the closed regions contain tangles
- Strategy iteration: “best response” stays in tangles
- Progress measures: slowly increase value until an escape is taken

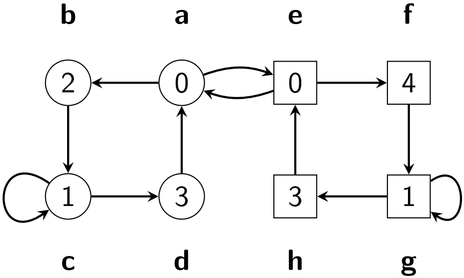
Often algorithms explore the same tangle many times!

Distractions



- The entire game is won by player Even.
  Determine the winning strategy?!
- A random vertex inside the winning region?
  If you play from **c** to **c** you lose.
- Always play to the highest reachable even vertex?
  If you play from **a** to **b** you lose.
- Vertex **b** is a distraction for player Even

Distractions



- To solve the above game, you need to avoid the distractions.
- Example: tangle learning:
  1. First round: tangle $\{\,\mathbf{c}\,\}$ (attracts distraction **b**)
  2. Second round: tangle $\{\,\mathbf{a},\mathbf{e}\,\}$ (attracts distraction **h**)
  3. Third round: tangle $\{\,\mathbf{g}\,\}$ (dominion)

Distractions

Intuition

A distraction for α is a "high value vertex" v with an α-priority that player $\overline{\alpha}$ can win if player α always tries to visit it. Some distractions are actually won by player α as long as some distracted vertices avoid the distraction.

Definition

A distraction for player α is a vertex v with an α-priority p, such that if player α always plays to reach v along paths of priorities $\leq p$, then player $\overline{\alpha}$ wins v and all vertices that reach v.
(Along these paths to v, player $\overline{\alpha}$ either must or chooses to play to v.)

Underlying structure

If vertex v is a distraction for player α, then player $\overline{\alpha}$ can attract vertex v (directly or via tangles) to either higher $\overline{\alpha}$-priority vertices, or to an $\overline{\alpha}$-dominion.

Distractions

Distraction

A distraction for player α is a vertex v with an α-priority p, such that if player α always plays to reach v along paths of priorities $\leq p$, then player $\overline{\alpha}$ wins v and all vertices that reach v.

Two cases

1. Player $\overline{\alpha}$ can attract v to vertices with higher $\overline{\alpha}$-priorities.
   Every play that infinitely often visits v, also infinitely often visits a vertex with a higher $\overline{\alpha}$-priority (or stays in a losing tangle).
2. Player $\overline{\alpha}$ can attract v to an $\overline{\alpha}$-dominion.
   Every play that visits v reaches the $\overline{\alpha}$-dominion.

Distractions

Distraction

A distraction for α is a high value vertex v with an α-priority that player $\overline{\alpha}$ can win if player α always tries to visit it.

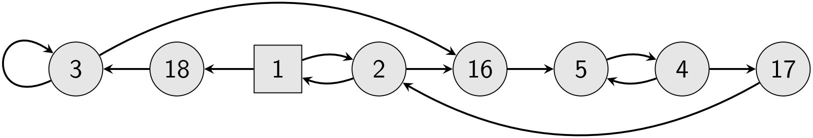
Types

- Trivial distractions
  The opponent attract v to higher $\overline{\alpha}$-priority vertices directly
- Non-trivial distractions
  The opponent attracts v to higher $\overline{\alpha}$-priority vertices via tangles
- Devious distractions
  A distraction for α that is actually in the winning region of α

  Player α must *sometimes* avoid v.
- Fatal distractions

  Player α must *always* avoid v.

Distractions



- **3** is a trivial distraction
- **17** is a non-trivial, fatal distraction
- **16** is a non-trivial, devious distraction

Distractions

Two fundamentally different approaches

All algorithms fundamentally must deal with distracting high value vertices.

Attractor-based algorithms identify distractions when the opponent attracts.

But we can keep attractor-based algorithms distracted for a very long time.
(Two Counters game)

Progress-based algorithms and strategy iteration ignore vertices that make no progress. Because "good' vertices along the path to the distraction get a higher value than the distraction. Eventually non-distractions (the "top" of tangles) obtain a higher value than the distractions.

But progress-based algorithms cannot see that the opponent attracts the vertex, even if the distraction is trivial.
(SPM/SI use different progressive values than quasipolynomial solvers)

Fixed point iteration

Core idea

- We can solve μ-calculus model checking...
  - with naive fixed point iteration
  - with "smarter" parity game solving algorithms
- We could also solve parity games with a fixed point iteration

Methods

- (**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!
- (**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!

Fixed point iteration

Distraction Fixed-point Iteration (DFI)

- "Update whether vertices are distractions by looking 1 step ahead"
- A vertex is a distraction if:
  - it has even priority and is won by Odd in 1 step
  - it has odd priority and is won by Even in 1 step
- Record "distraction sets" $Z_p \subseteq V_p$ $\qquad (V_p = \{v \mid \mathsf{pr}(v) = p\})$
- Initially assume no vertex is a distraction.
- 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
- In fixed point: Z contains exactly all fatal distractions

Fixed point iteration

Given some set of distracting vertices $Z = Z_0 \cup Z_1 \cup \cdots \cup Z_d$,

$$\mathsf{winner}(v, Z) \quad := \quad \begin{cases} \mathsf{pr}(v) \bmod 2 & v \notin Z \\ 1 - (\mathsf{pr}(v) \bmod 2) & v \in Z \end{cases}$$

$$\mathsf{onestep}(v, Z) \quad := \quad \begin{cases} 0 & v \in V_{\bigcirc} \wedge \exists u \in E(v) : \mathsf{winner}(u, Z) = 0 \\ 1 & v \in V_{\bigcirc} \wedge \forall u \in E(v) : \mathsf{winner}(u, Z) = 1 \\ 1 & v \in V_{\square} \wedge \exists u \in E(v) : \mathsf{winner}(u, Z) = 1 \\ 0 & v \in V_{\square} \wedge \forall u \in E(v) : \mathsf{winner}(u, Z) = 0 \end{cases}$$

$$\mathsf{OnestepDistraction}(Z) := \{v \mid \mathsf{onestep}(v, Z) \neq \mathsf{pr}(v) \bmod 2\}$$

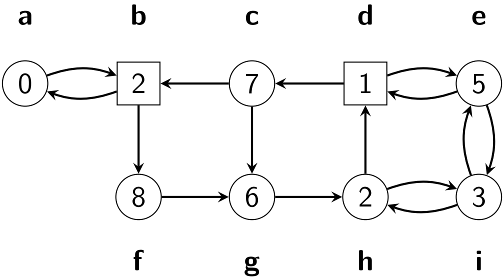
$$\mu Z_d \ldots \mu Z_1 \ . \ \mu Z_0 \ . \ \mathsf{OnestepDistraction}\big(\textstyle\bigvee_{p=0}^{d}(V_p \wedge Z_p)\big)\big)$$

Fixed point iteration

def fpi($\Game$):
    $Z \leftarrow \emptyset$ // start with no distractions
    $p \leftarrow 0$ // start with lowest priority
    while $p \leq d$ **:** // while $\leq$ highest priority
        $\alpha \leftarrow p \bmod 2$ // current parity
        $Y \leftarrow \{v \in V_p \setminus Z \mid \mathsf{onestep}(v, Z) \neq \alpha\}$ // new distractions
        if $Y \neq \emptyset$ **:**
            $Z \leftarrow Z \cup Y$ // update current fixed point Z_p
            $Z \leftarrow Z \setminus \{v \mid \mathsf{pr}(v) < p\}$ // reset all lower fixed points
            $p \leftarrow 0$ // restart with lowest priority
        else:
            $p \leftarrow p + 1$ // fixed point, continue higher
    return $W_\bigcirc, W_\Box$ where $W_\bigcirc \leftarrow \{v \mid \mathsf{winner}(v, Z) = 0\}$, $W_\Box \leftarrow V \setminus W_\bigcirc$

Note: algorithm does not give a strategy!

Fixed point iteration



a? d? b? **h!** a? d? b? i? e? **g!** (reset **h**)
a? d? b? **h!** a? d? b? i? e? **c!** (reset **h**, **g**)
a? d? b? **h!** a? d? b? i? e? **g!** (reset **h**)
a? d? b? **h!** a? d? b? i? e? **f!** (reset **h**, **g**, **c**)
a? d? **b! h!** **a!** d? i? e? **g!** (reset **b**, **h**, **a**)
a? d? **b! h!** **a!** d? i? e? c?

Final: $Z := \{\mathbf{a}, \mathbf{b}, \mathbf{h}, \mathbf{g}, \mathbf{f}\}$

Proof

What is a correct solution?

We have two winning regions $W_\bigcirc$ and $W_\Box$, such that:

1. Player α must win in W_α
   That is: there must exist a winning strategy $\sigma_\alpha \colon (V_\alpha \cap W_\alpha) \to W_\alpha$, such that all cycles in $W_\alpha[\sigma_\alpha]$ are won by α
2. Player $\overline{\alpha}$ must not be able to leave W_α

Approach

Prove by induction that the property holds after each fixed point Z_p.
Then it is true after Z_d (entire game).

Proof

Proof by induction

- Lemma is trivially true for the empty game.
- Assume it is true after computing Z_{p-1}.
  - So there exist winning strategies $\sigma_\bigcirc$ and $\sigma_\Box$ for the subgame $< p$
- Prove for Z_p by induction on each step of the fixed point computation by constructing strategies $\sigma_\bigcirc$ and $\sigma_\Box$ for the subgame $\leq p$

Proof

Lemma

After computing the fixpoint of Z_p, player $\alpha \in \{\circ, \square\}$ has a winning region $W_\alpha \equiv \mathit{Won}(\alpha)_{\leq p}$ and a strategy σ_α for all $v \in V_\alpha \cap W_\alpha$, such that

- *player α never plays from W_α to $\mathit{Won}(\overline{\alpha})$*
- *player $\overline{\alpha}$ cannot play from W_α to $\mathit{Won}(\overline{\alpha})$*
- *all cycles consistent with σ_α in W_α are won by α*

Proof

Implementation

Implementation notes

- Idea: do not recompute whether vertices won by $\overline{\alpha}$ in the subgame $< p$ are a distraction
  - Vertices in $Z_0 \cdots Z_{p-1}$ of player α stay distractions
  - Vertices *not* in $Z_0 \cdots Z_{p-1}$ of player α stay non-distractions
- "Frozen" sets $F_0 \cdots F_d$ record which vertices are frozen
- After finishing a fixed-point Z_p, set $F_p := \emptyset$.
- Now obtaining the winning strategy is trivial.

Evaluation

	equivalence checking		model-checking		reactive synthesis	
priorities	2		1–4		3–12	
count	216		313		223	
	mean	**max**	**mean**	**max**	**mean**	**max**
# vertices	3,288,890	40,556,396	866,289	27,876,961	921,484	31,457,288
# edges	10,121,422	167,527,601	2,904,500	80,830,465	1,693,544	59,978,691
outdegree	2.35	5.09	2.75	6.14	1.68	2.00

Table: Statistics of the three benchmark sets used for the empirical evaluation.

Dataset	**with preprocessing**					**without preprocessing**				
	`fpi`	`zlk`	`pp`	`tl`	`si`	`fpi`	`zlk`	`pp`	`tl`	`si`
equivalence	401	**381**	389	455	6218	970	470	444	570	19568
model-checking	**59**	73	82	166	292	156	79	93	182	2045
synthesis	62	**52**	57	59	158	64	**51**	70	67	175

Table: Cumulative time in seconds (average of five runs) spent to solve all games in each set of benchmarks, with a timeout of 1800 seconds. We record 1800 seconds when the computation timed out.

Summary

- Distraction Fixed-point Iteration: compute via nested fixed-points which vertices are distractions, then infer the winning regions
- Using frozen sets, we trivially obtain the winning strategy
- `DFI`, `BFL`, `APT` are equivalent (not in the slides)
- `DFI` is simple to implement
- Very nice benchmark results: `DFI` is surprisingly fast for model-checking

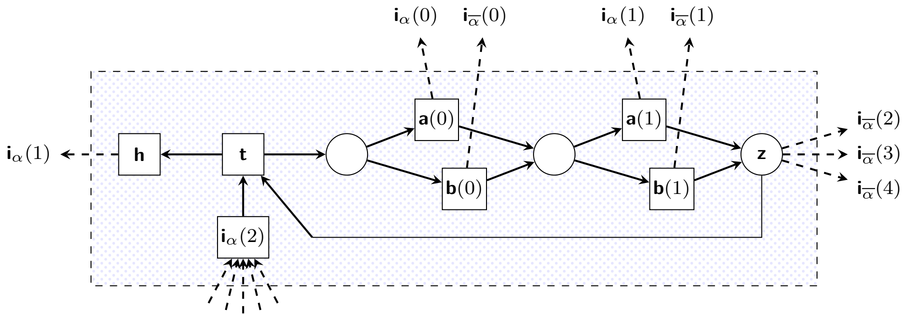
Overview

A Parity Game Tale of Two Counters

- Context
  - Broad goal: develop polynomial parity games solution
  - Using concepts like tangles, distractions
  - Many types of algorithms that deal with distractions differently
  - If not polynomial, deepen understanding with *strong* counterexamples
- Contributions
  - Parameterized parity game family for:
    - Recursive algorithm (Zielonka)
    - Priority promotion & variations
    - Tangle learning

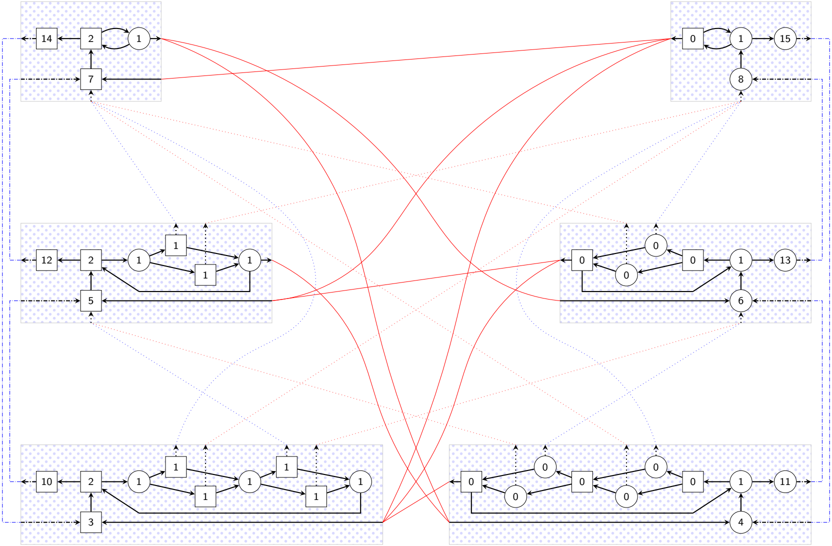
    All attractor-based algorithms that recognize a distraction by finding that the opponent can attract the distraction
  - Based on two interleaved binary counters of N bits

A bit in the game



- $\mathbf{i}_\alpha$ has opponent's ($\overline{\alpha}$'s) priority, rest α's priority
- If the tangle is learned, $\mathbf{i}_\alpha$ is attracted to a higher α priority
- The tangle is not learned until **z** plays to **t** instead of the distractions
  That is: first learn lower bits of the opponent counter
- The tangle is "disabled" when higher bits ()of opponent) are set.
  That is: learning a bit resets all lower bits of the opponent counter

Example 3-bit Game



Summary

Reflections

- Defeats all algorithms that rely only on opponent attraction to identify distractions.
- Until opponent attraction, the player is distracted and cannot find the distracted tangles.

- Other method: play to the best path, not just the best endpoint
  This is like progressive path values in progress measures
- "Recursive tangle learning" and "progressive tangle learning" (unpublished) use this method to identify distractions and can even use both methods to find distracted tangles.

Summary

Reflections

- Defeats all algorithms that rely only on opponent attraction to identify distractions.
- Until opponent attraction, the player is distracted and cannot find the distracted tangles.

- Other method: play to the best path, not just the best endpoint
  This is like progressive path values in progress measures
- "Recursive tangle learning" and "progressive tangle learning" (unpublished) use this method to identify distractions and can even use both methods to find distracted tangles.

  BUT VULNERABLE! (to a different version of the TC game)!

Tangles and distractions

Quasi-polynomial algorithms and distractions

- Look at “how many times α dominates beyond this vertex”
- Similar to “distance to opponent region”
- Also a path-progressive measure, but stronger

Shortcomings of the quasi-polynomial progress-based algorithms

- They are not aware of tangles (!!)
- They are short-sighted, not aware of even trivial distractions.
  (they do not identify distractions based on opponent attraction)
- They seem to involve a LOT of repetition
- They solve the entire game, instead of returning a dominion.
- **So there is still hope!**