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 $\mu$-calculus translation of Zielonka’s recursive algorithm
    • Encoding to $\mu$-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: $\exists, \lor, \Diamond$, player 1: $\forall, \land, \Box$)
- 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
Why do we want to solve parity games?

- Expressive power of nested least and greatest fixpoint operators
- Linear equivalence with modal $\mu$-calculus model-checking (CTL, LTL, CTL*, ...)
- Backend for LTL model checking and LTL synthesis
  Important industrial applications (PSL, SVA)
Why do we want to solve parity games?

- Expressive power of nested least and greatest fixpoint operators
- Linear equivalence with modal $\mu$-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 $NP \cap co-NP$ and in $UP \cap co-UP$
- Hot topic! Recently new quasi-polynomial algorithms and even quasi-polynomial lower bounds for many algorithms!
### Parity Games

#### (Incomplete list of) algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time Complexity</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>McNaughton/Zielonka</td>
<td>$O(e \cdot n^d)$, $O(2^n)$</td>
<td>1998</td>
</tr>
<tr>
<td>Small Progress Measures</td>
<td>$O(d \cdot e \cdot (n/d)^{d/2})$</td>
<td>1998</td>
</tr>
<tr>
<td>Strategy Improvement</td>
<td>$O(n \cdot e \cdot 2^e)$</td>
<td>2000</td>
</tr>
<tr>
<td>Dominion Decomposition</td>
<td>$O(n\sqrt{n})$</td>
<td>2006</td>
</tr>
<tr>
<td>Big Step</td>
<td>$O(e \cdot n^{d/3})$</td>
<td>2007</td>
</tr>
<tr>
<td><strong>Fixed Point: APT, BFL, DFI</strong></td>
<td>$O(n^d)$</td>
<td>2014 – 2019</td>
</tr>
<tr>
<td>Priority Promotion</td>
<td>$\Omega(2\sqrt{n})$</td>
<td>2016</td>
</tr>
<tr>
<td>Quasi-Polynomial (several)</td>
<td>$O(n^{6+\log d})$</td>
<td>2016 – 2018</td>
</tr>
<tr>
<td>Tangle Learning</td>
<td>$\Omega(2\sqrt{n})$</td>
<td>2018</td>
</tr>
<tr>
<td>Recursive Tangle Learning</td>
<td>$\Omega(2\sqrt{n})$</td>
<td>ongoing</td>
</tr>
<tr>
<td>Progressive Tangle Learning</td>
<td>$\Omega(2\sqrt{n}) (!)$</td>
<td>ongoing</td>
</tr>
</tbody>
</table>
Parity Games

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

The play $\pi$: a
A parity game is played on a directed graph.

Two players: Even \(\bigcirc\) and Odd \(\Box\).

The players move a token along the edges of the graph.

Each vertex is owned by one player who chooses a successor.

The play \(\pi\): \(a\,b\).
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 $\pi$: a b d
Parity Games

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

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

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

The play $\pi$: $abdec e$
A parity game is played on a directed graph.

Two players: Even $\bigcirc$ and Odd $\square$

The players move a token along the edges of the graph.

Each vertex is owned by one player who chooses a 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 □
• The players move a token along the edges of the graph
• Each vertex is owned by one player who chooses a successor

The play $\pi$: a b d e c e c b
A parity game is played on a directed graph.

Two players: Even \( \bigcirc \) and Odd \( \square \)

The players move a token along the edges of the graph.

Each vertex is owned by one player who chooses a 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 $\bigcirc$ and Odd $\square$
- The players move a token along the edges of the graph
- Each vertex is owned by one player who chooses a successor

The play $\pi$: a b d e c e c b d a
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 \( \bigcirc \) and Odd \( \square \)
- 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
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?
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.
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?
A parity game is played on a directed graph. Two players: Even $\bigcirc$ and Odd $\square$. The players move a token along the edges of the graph. Each vertex is owned by one player who chooses a successor. The highest priority seen infinitely often determines the winner. Player Odd wins all vertices with strategy $\{d \rightarrow e\}$. 
Parity Games

Basic facts of parity games

- Partition into $W_\circ$ (won by Even) and $W_\square$ (won by Odd)
- The winner $\alpha$ has a positional strategy $\sigma_\alpha : V_\alpha \to V$
- All plays consistent in $W_\alpha$ with $\sigma_\alpha$ are won by $\alpha$
  That is: player $\alpha$ wins all cycles in $W_\alpha[\sigma_\alpha]$

Solving a parity game

- Compute the winning regions $W_\circ$ and $W_\square$
- Compute the winning strategies $\sigma_\circ$ and $\sigma_\square$
Parity Games

Attractor computation

• “Reachability but in a game”

• Compute all vertices from which player $\alpha \in \{\lozenge, \square\}$ can ensure arrival in a given target set

• Start with the target set $Z$, then iteratively add:
  • all vertices of $\alpha$ that can play to $Z$
  • all vertices of $\overline{\alpha}$ that can only play to $Z$
Example of attractor computation

Computing the \( \Box \)-attractor to \( a \)

Initial set: \( \{ a \} \)
Can attract: \( d \) but not \( b \)
Example of attractor computation

Computing the $\Box$-attractor to $a$

Current set: $\{a, d\}$
Can attract: $b$ but not $e$
Example of attractor computation

Computing the □-attractor to **a**

Current set: \{ **a**, **b**, **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
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.
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
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.


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.
Tangles

Definition

A tangle is

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

such that

- the induced subgame \( \mathcal{G}[U, \sigma] \) is strongly connected
- player \( \alpha \) wins all cycles in the induced subgame \( \mathcal{G}[U, \sigma] \)
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 $\alpha$ 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
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!
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
• To solve the above game, you need to avoid the distractions.

• Example: tangle learning:
  1 First round: tangle \{c\} (attracts distraction b)
  2 Second round: tangle \{a, e\} (attracts distraction h)
  3 Third round: tangle \{g\} (dominion)
Intuition

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

Definition

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

Two cases

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

2. **Player $\bar{\alpha}$ can attract $v$ to an $\bar{\alpha}$-dominion.**
   Every play that visits $v$ reaches the $\bar{\alpha}$-dominion.
Distractions

Distraction
A distraction for \( \alpha \) is a high value vertex \( v \) with an \( \alpha \)-priority that player \( \overline{\alpha} \) can win if player \( \alpha \) 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 \( \alpha \) that is actually in the winning region of \( \alpha \)
  Player \( \alpha \) must *sometimes* avoid \( v \).

- **Fatal** distractions
  Player \( \alpha \) must *always* avoid \( v \).
• 3 is a trivial distraction
• 17 is a non-trivial, fatal distraction
• 16 is a non-trivial, devious distraction
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 $\mu$-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 $\mu$-calculus translation of Zielonka’s recursive algorithm
  - Encoding to $\mu$-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**” $Z_p \subseteq V_p$ ($V_p = \{v \mid \text{pr}(v) = p\}$)
- 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 \( Z = Z_0 \cup Z_1 \cup \cdots \cup Z_d \),

\[
\text{winner}(v, Z) := \begin{cases} 
\text{pr}(v) \mod 2 & v \notin Z \\
1 - (\text{pr}(v) \mod 2) & v \in Z 
\end{cases}
\]

\[
\text{onestep}(v, Z) := \begin{cases} 
0 & v \in \mathcal{V}_0 \land \exists u \in E(v) : \text{winner}(u, Z) = 0 \\
1 & v \in \mathcal{V}_0 \land \forall u \in E(v) : \text{winner}(u, Z) = 1 \\
1 & v \in \mathcal{V}_1 \land \exists u \in E(v) : \text{winner}(u, Z) = 1 \\
0 & v \in \mathcal{V}_1 \land \forall u \in E(v) : \text{winner}(u, Z) = 0 
\end{cases}
\]

\[
\text{OnestepDistraction}(Z) := \{ v \mid \text{onestep}(v, Z) \neq \text{pr}(v) \mod 2 \}
\]

\[
\mu Z_d \ldots \mu Z_1 \cdot \mu Z_0 \cdot \text{OnestepDistraction}(\bigvee_{p=0}^{d}(V_p \land Z_p))
\]
def fpi(∅):
    Z ← ∅  // start with no distractions
    p ← 0  // start with lowest priority
    while p ≤ d :  // while ≤ highest priority
        α ← p mod 2  // current parity
        Y ← {v ∈ V_p \ Z | onestep(v, Z) ≠ α}  // new distractions
        if Y ≠ ∅ :
            Z ← Z ∪ Y  // update current fixed point Z_p
            Z ← Z \ {v | pr(v) < p}  // reset all lower fixed points
            p ← 0  // restart with lowest priority
        else:
            p ← p + 1  // fixed point, continue higher
    return W_⊙, W_□ where W_⊙ ← {v | winner(v, Z) = 0}, W_□ ← V \ W_⊙

Note: algorithm does not give a strategy!
Fixed point iteration

$$Z := \{a, b, h, g, f\}$$
What is a correct solution?

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

1. Player $\alpha$ must win in $W_\alpha$
   That is: there must exist a winning strategy $\sigma_\alpha : (V_\alpha \cap W_\alpha) \rightarrow W_\alpha$, such that all cycles in $W_\alpha[\sigma_\alpha]$ are won by $\alpha$

2. Player $\bar{\alpha}$ must not be able to leave $W_\alpha$

Approach

Prove by induction that the property holds after each fixed point $Z_p$. Then it is true after $Z_d$ (entire game).
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 $\sigma_\square$ for the subgame $< p$
- Prove for $Z_p$ by induction on each step of the fixed point computation by constructing strategies $\sigma_\bigcirc$ and $\sigma_\square$ for the subgame $\leq p$
Lemma

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

- player $\alpha$ never plays from $W_{\alpha}$ to $\text{Won}(\overline{\alpha})$
- player $\overline{\alpha}$ cannot play from $W_{\alpha}$ to $\text{Won}(\overline{\alpha})$
- all cycles consistent with $\sigma_{\alpha}$ in $W_{\alpha}$ are won by $\alpha$
Implementation notes

- Idea: do not recompute whether vertices won by $\bar{\alpha}$ in the subgame $< p$ are a distraction
  - Vertices in $Z_0 \cdots Z_{p-1}$ of player $\alpha$ stay distractions
  - Vertices not in $Z_0 \cdots Z_{p-1}$ of player $\alpha$ 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

<table>
<thead>
<tr>
<th>Priorities count</th>
<th>Equivalence Checking</th>
<th>Model-Checking</th>
<th>Reactive Synthesis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>1–4</td>
<td>3–12</td>
</tr>
<tr>
<td></td>
<td>216</td>
<td>313</td>
<td>223</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># Vertices</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3,288,890</td>
<td>40,556,396</td>
<td>866,289</td>
<td>27,876,961</td>
<td>921,484</td>
<td>31,457,288</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># Edges</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10,121,422</td>
<td>167,527,601</td>
<td>2,904,500</td>
<td>80,830,465</td>
<td>1,693,544</td>
<td>59,978,691</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outdegree</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.35</td>
<td>5.09</td>
<td>2.75</td>
<td>6.14</td>
<td>1.68</td>
<td>2.00</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>With Preprocessing</th>
<th>Without Preprocessing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fpi</td>
<td>zlk</td>
</tr>
<tr>
<td>Equivalence</td>
<td>401</td>
<td><strong>381</strong></td>
</tr>
<tr>
<td>Model-Checking</td>
<td><strong>59</strong></td>
<td>73</td>
</tr>
<tr>
<td>Synthesis</td>
<td>62</td>
<td><strong>52</strong></td>
</tr>
</tbody>
</table>

**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.
• 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
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
• $i_\alpha$ has opponent’s ($\overline{\alpha}$’s) priority, rest $\alpha$’s priority

• If the tangle is learned, $i_\alpha$ is attracted to a higher $\alpha$ 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 ($\overline{\alpha}$) of opponent) are set.
  That is: learning a bit resets all lower bits of the opponent counter
Example 3-bit Game
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.
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 $\alpha$ 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!