Universal algorithms for parity games and nested fixpoints

ANR Delta meeting

Marcin Jurdziński¹, Rémi Morvan², K. S. Thejaswini¹

June 28, 2021, in Paris!

¹University of Warwick
²École normale supérieure Paris-Saclay
\( G = \langle V, E, V_{\text{Even}}, V_{\text{Odd}}, \pi : V \rightarrow [0, d] \rangle \)
Parity games

\[ G = \langle V, E, V_{\text{Even}}, V_{\text{Odd}}, \pi : V \rightarrow [0, d] \rangle \]
Parity games

\[ G = \langle V, E, V_{\text{Even}}, V_{\text{Odd}}, \pi : V \rightarrow \mathbb{N} \rangle \]
• Play: \((v_i)_{i \in \mathbb{N}}\) s.t. \(\forall i, (v_i, v_{i+1}) \in E\).
Plays

- Play: $(v_i)_{i \in \mathbb{N}}$ s.t. $\forall i, (v_i, v_{i+1}) \in E.$
Plays

- Play: \((v_i)_{i \in \mathbb{N}}\) s.t. \(\forall i, (v_i, v_{i+1}) \in E\).
- Even wins \((v_i)_{i \in \mathbb{N}}\) iff the maximal priority occurring infinitely often is even.
Parity games
McNaughton-Zielonka
Universal algorithm
Attractor decomposition
Conclusion

Plays

- Play: \((v_i)_{i \in \mathbb{N}}\) s.t. \(\forall i, (v_i, v_{i+1}) \in E\).
- Even wins \((v_i)_{i \in \mathbb{N}}\) iff the maximal priority occurring infinitely often is even.

R. Morvan
• Memoryless strategy for Odd: \( \sigma : V_{Odd} \rightarrow V \) s.t. \((v, \sigma(v)) \in E\) for every vertex \(v\).
- Memoryless strategy for Odd: $\sigma : V_{Odd} \to V$ s.t. $(v, \sigma(v)) \in E$ for every vertex $v$. 
Strategies

• Memoryless strategy for Odd: $\sigma : V_{\text{Odd}} \rightarrow V$ s.t. $(v, \sigma(v)) \in E$ for every vertex $v$.

• A play $(v_i)_{i \in \mathbb{N}}$ is consistent with $\sigma$ if $v_{i+1} = \sigma(v_i)$ whenever $\sigma(v_i)$ is defined.
Strategies

- Memoryless strategy for Odd: \( \sigma : V_{\text{Odd}} \rightarrow V \) s.t. \((v, \sigma(v)) \in E\) for every vertex \(v\).

- A play \((v_i)_{i \in \mathbb{N}}\) is consistent with \(\sigma\) if \(v_{i+1} = \sigma(v_i)\) whenever \(\sigma(v_i)\) is defined.
Strategies

- **Memoryless strategy for Odd**: $\sigma : V_{\text{Odd}} \rightarrow V$ s.t. $(v, \sigma(v)) \in E$ for every vertex $v$.

- A play $(v_i)_{i \in \mathbb{N}}$ is consistent with $\sigma$ if $v_{i+1} = \sigma(v_i)$ whenever $\sigma(v_i)$ is defined.
Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy $\sigma$ such that every play consistent with $\sigma$ and starting at $v_0$ is winning for her.
Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy $\sigma$ such that every play consistent with $\sigma$ and starting at $v_0$ is winning for her.
Memoryless determinacy

Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy $\sigma$ such that every play consistent with $\sigma$ and starting at $v_0$ is winning for her.
Memoryless determinacy

Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy $\sigma$ such that every play consistent with $\sigma$ and starting at $v_0$ is winning for her.
Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy $\sigma$ such that every play consistent with $\sigma$ and starting at $v_0$ is winning for her.
Theorem: From every vertex \( v_0 \in V \), one of the two players has a memoryless strategy \( \sigma \) such that every play consistent with \( \sigma \) and starting at \( v_0 \) is winning for her.
Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy $\sigma$ such that every play consistent with $\sigma$ and starting at $v_0$ is winning for her.
Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy $\sigma$ such that every play consistent with $\sigma$ and starting at $v_0$ is winning for her.
Solving parity games

**Solving Parity Games:**

Inputs: \( G \): parity game,

\( v_0 \in V^G \): vertex.

Question: Can player Even win from \( v_0 \) in \( G \)?
Solving parity games

**SOLVING PARITY GAMES:**

**Inputs:**
- $G$: parity game,
- $v_0 \in V^G$: vertex.

**Question:** Can player Even win from $v_0$ in $G$?

- In $\mathsf{NP} \cap \mathsf{coNP}$. 

---

R. Morvan
Solving parity games

**SOLVING PARITY GAMES:**

**Inputs:** \( G \): parity game,
\( v_0 \in V^G \): vertex.

**Question:** Can player Even win from \( v_0 \) in \( G \)?

- In \( \text{NP} \cap \text{coNP} \).
- Believed to be in \( \text{P} \)…
Solving parity games

Solving Parity Games:
Inputs: \( G \): parity game,
\( v_0 \in V^G \): vertex.
Question: Can player Even win from \( v_0 \) in \( G \)?

- In \( \text{NP} \cap \text{coNP} \).
- Believed to be in \( \text{P} \)…
- Best known upper bound: quasipolynomial time \( O(n^{\log(d)}) \) [’17 Calude-Jain-Khoussainov-Li-Stephan]
A recursive algorithm?

How to compute the winning vertices of Odd?
A recursive algorithm?

How to compute the winning vertices of Odd?

1. Identify a “small” winning set $U$ for Odd.
A recursive algorithm?

How to compute the winning vertices of Odd?

1. Identify a “small” winning set $U$ for Odd. How???
A recursive algorithm?

How to compute the winning vertices of Odd?

1. Identify a “small” winning set $U$ for Odd. How???
A recursive algorithm?

How to compute the winning vertices of Odd?

1. Identify a “small” winning set $U$ for Odd. How???

2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}^G_{\text{Odd}}(U)$. 
A recursive algorithm?

How to compute the winning vertices of Odd?

1. Identify a “small” winning set $U$ for Odd. How???

2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}^G_{\text{Odd}}(U)$.
How to compute the winning vertices of Odd?

1. Identify a “small” winning set $U$ for Odd. How???

2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}_{\text{Odd}}^G(U)$.

3. Iterate: compute Odd’s winning vertices in $G \setminus \text{Attr}_{\text{Odd}}^G(U)$.
A recursive algorithm?

How to compute the winning vertices of Odd?

1. Identify a “small” winning set $U$ for Odd. How???

2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}^G_{\text{Odd}}(U)$.

3. Iterate: compute Odd’s winning vertices in $G \setminus \text{Attr}^G_{\text{Odd}}(U)$. If you don’t find any, stop.
McNaughton-Zielonka’s algorithm: motivations

How can one identify a “small” winning set $U$ for Odd?
McNaughton-Zielonka’s algorithm: motivations

How can one identify a “small” winning set $U$ for Odd?

McNaughton-Zielonka’s answer:
McNaughton-Zielonka’s algorithm: motivations

How can one identify a “small” winning set $U$ for Odd?

McNaughton-Zielonka’s answer:

- $d$ greatest priority; wlog. $d$ is even.
McNaughton-Zielonka’s algorithm: motivations

How can one identify a “small” winning set $U$ for Odd?

**McNaughton-Zielonka’s answer:**

- $d$ greatest priority; wlog. $d$ is even.
- If Odd wins from $v$, how many vertices of priority $d$ will we see?
McNaughton-Zielonka’s algorithm: motivations

How can one identify a “small” winning set $U$ for Odd?

McNaughton-Zielonka’s answer:

- $d$ greatest priority; wlog. $d$ is even.
- If Odd wins from $v$, how many vertices of priority $d$ will we see?
  - none
  - at least one, but finitely many
  - infinitely many
McNaughton-Zielonka’s algorithm: motivations

How can one identify a “small” winning set $U$ for Odd?

McNaughton-Zielonka’s answer:

- $d$ greatest priority ; wlog. $d$ is even.
- If Odd wins from $v$, how many vertices of priority $d$ will we see?
  - none
  - at least one, but finitely many
  - infinitely many

How to compute the set of $v$ s.t. Odd can win from $v$ without ever seeing a vertex of priority $d$?
McNaughton-Zielonka’s algorithm: motivations

How can one identify a “small” winning set $U$ for Odd?

McNaughton-Zielonka’s answer:

- $d$ greatest priority; wlog. $d$ is even.
- If Odd wins from $v$, how many vertices of priority $d$ will we see?
  - none ← easy to identify!
  - at least one, but finitely many
  - infinitely many

How to compute the set of $v$ s.t. Odd can win from $v$ without ever seeing a vertex of priority $d$? It is the set of winning vertices for Odd in the game

$$G \setminus \text{Attr}^G_{\text{Even}}(\pi^{-1}[d]).$$
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. Identify a “small” winning set $U$ for Odd.
2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}^G_{\text{Odd}}(U)$.
3. Iterate: compute Odd’s winning vertices in $G \setminus \text{Attr}^G_{\text{Odd}}(U)$. If you don’t find any, stop.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. $U =$ vertices from which Odd can win while avoiding $d$.
2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}^G_{\text{Odd}}(U)$.
3. Iterate: compute Odd’s winning vertices in $G \setminus \text{Attr}^G_{\text{Odd}}(U)$. If you don’t find any, stop.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. \( U = \) vertices from which Odd can win while avoiding \( d \).
2. Consider the set of vertices from which Odd can force the play to reach \( U \), denoted \( \text{Attr}^G_{\text{Odd}}(U) \).
3. Iterate: compute Odd’s winning vertices in \( G \setminus \text{Attr}^G_{\text{Odd}}(U) \). If you don’t find any, stop.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. \( U \) = vertices from which Odd can win while avoiding \( d \).
2. Consider the set of vertices from which Odd can force the play to reach \( U \), denoted \( \text{Attr}^G_{\text{Odd}}(U) \).
3. Iterate: compute Odd’s winning vertices in \( G \setminus \text{Attr}^G_{\text{Odd}}(U) \). If you don’t find any, stop.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. \( U \) = vertices from which Odd can win while avoiding \( d \).
2. Consider the set of vertices from which Odd can force the play to reach \( U \), denoted \( \text{Attr}^G_{\text{Odd}}(U) \).
3. Iterate: compute Odd’s winning vertices in \( G \setminus \text{Attr}^G_{\text{Odd}}(U) \). If you don’t find any, stop.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. \( U \) = vertices from which Odd can win while avoiding \( d \).
2. Consider the set of vertices from which Odd can force the play to reach \( U \), denoted \( \text{Attr}^G_{\text{Odd}}(U) \).
3. Iterate: compute Odd’s winning vertices in \( G \setminus \text{Attr}^G_{\text{Odd}}(U) \). If you don’t find any, stop.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. \( U \) = vertices from which Odd can win while avoiding \( d \).
2. Consider the set of vertices from which Odd can force the play to reach \( U \), denoted \( \text{Attr}^G_{\text{Odd}}(U) \).
3. Iterate: compute Odd’s winning vertices in \( G \setminus \text{Attr}^G_{\text{Odd}}(U) \). If you don’t find any, stop.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. $U =$ vertices from which Odd can win while avoiding $d$.
2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}^G_{\text{Odd}}(U)$.
3. Iterate: compute Odd’s winning vertices in $G \setminus \text{Attr}^G_{\text{Odd}}(U)$. If you don’t find any, stop.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. $U$ = vertices from which Odd can win while avoiding $d$.
2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}^G_{\text{Odd}}(U)$.
3. Iterate: compute Odd’s winning vertices in $G \setminus \text{Attr}^G_{\text{Odd}}(U)$. If you don’t find any, stop.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. \( U \) = vertices from which Odd can win while avoiding \( d \).
2. Consider the set of vertices from which Odd can force the play to reach \( U \), denoted \( \text{Attr}^G_{\text{Odd}}(U) \).
3. Iterate: compute Odd’s winning vertices in \( G \setminus \text{Attr}^G_{\text{Odd}}(U) \). If you don’t find any, stop.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. $\mathbf{U} =$ vertices from which Odd can win while avoiding $d$.
2. Consider the set of vertices from which Odd can force the play to reach $\mathbf{U}$, denoted $\text{Attr}^G_{\text{Odd}}(\mathbf{U})$.
3. Iterate: compute Odd’s winning vertices in $\mathcal{G} \setminus \text{Attr}^G_{\text{Odd}}(\mathbf{U})$. If you don’t find any, stop.
McNaughton-Zielonka’s algorithm

Recursive algorithm. At each call: fewer priorities or fewer vertices.

1. $U$ = vertices from which Odd can win while avoiding $d$.
2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}_{\text{Odd}}^{G}(U)$.
3. Iterate: compute Odd’s winning vertices in $G \setminus \text{Attr}_{\text{Odd}}^{G}(U)$. If you don’t find any, stop.
McNaughton-Zielonka’s algorithm: correctness

• Correctness: by construction.
McNaughton-Zielonka’s algorithm: correctness

- Correctness: by construction.
- Completeness: ???

Need to prove: "if there is no vertex from which Odd can win while avoiding $d$, then there is no winning vertex for Odd".

Recall that: "If Odd wins from $v$, how many vertices of priority $d$ will we see?
- none: easy to identify!
- at least one, but finitely many
- infinitely many"
McNaughton-Zielonka’s algorithm: correctness

- Correctness: by construction.
- Completeness: ????

Need to prove: “if there is no vertex from which Odd can win while avoiding $d$, then there is no winning vertex for Odd”.
McNaughton-Zielonka’s algorithm: correctness

- Correctness: by construction.
- Completeness: ???

Need to prove: “if there is no vertex from which Odd can win while avoiding \( d \), then there is no winning vertex for Odd”.

Recall that: “If Odd wins from \( v \), how many vertices of priority \( d \) will we see?

- none \( \leftarrow \) easy to identify!
- at least one, but finitely many
- infinitely many”
McNaughton-Zielonka’s algorithm: correctness

- Correctness: by construction.
- Completeness: ?? by construction!

Need to prove: “if there is no vertex from which Odd can win while avoiding \( d \), then there is no winning vertex for Odd”. Recall that: “If Odd wins from \( v \), how many vertices of priority \( d \) will we see?
  - none \( \leftarrow \) easy to identify!
  - at least one, but finitely many
  - infinitely many”
1. \( U \) = vertices from which Odd can win while avoiding \( d \).

2. Consider the set of vertices from which Odd can force the play to reach \( U \), denoted \( \text{Attr}^G_{\text{Odd}}(U) \).

3. Iterate: compute the Odd’s winning vertices in \( G \setminus \text{Attr}^G_{\text{Odd}}(U) \). If you don’t find any, stop.
McNaughton-Zielonka’s algorithm: complexity

1. \( U \) = vertices from which Odd can win while avoiding \( d \). One recursive call with fewer priorities!

2. Consider the set of vertices from which Odd can force the play to reach \( U \), denoted \( \text{Attr}^G_{\text{Odd}}(U) \).

3. Iterate: compute the Odd’s winning vertices in \( G \setminus \text{Attr}^G_{\text{Odd}}(U) \). If you don’t find any, stop.
McNaughton-Zielonka’s algorithm: complexity

1. $U =$ vertices from which Odd can win while avoiding $d$. One recursive call with fewer priorities!

2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}_{\text{Odd}}^G(U)$. Computable in poly. time!

3. Iterate: compute the Odd’s winning vertices in $G \setminus \text{Attr}_{\text{Odd}}^G(U)$. If you don’t find any, stop.
McNaughton-Zielonka’s algorithm: complexity

1. \( U \) = vertices from which Odd can win while avoiding \( d \). One recursive call with fewer priorities!

2. Consider the set of vertices from which Odd can force the play to reach \( U \), denoted \( \text{Attr}^G_{\text{Odd}}(U) \). Computable in poly. time!

3. Iterate: compute the Odd’s winning vertices in \( G \setminus \text{Attr}^G_{\text{Odd}}(U) \). If you don’t find any, stop. How many times will we need to iterate?
McNaughton-Zielonka’s algorithm: complexity

1. \( U \) = vertices from which Odd can win while avoiding \( d \). One recursive call with fewer priorities!

2. Consider the set of vertices from which Odd can force the play to reach \( U \), denoted \( \text{Attr}^G_{\text{Odd}}(U) \). Computable in poly. time!

3. Iterate: compute the Odd’s winning vertices in \( G \setminus \text{Attr}^G_{\text{Odd}}(U) \). If you don’t find any, stop. How many times will we need to iterate? At most \( n = |V| \). 

\[ C_{n,d} = \text{n copies of } C_{n,d-1} \]
Universal algorithm

- Goal: solve a parity game whose priorities are $[0, d]$. 
Universal algorithm

- Goal: solve a parity game whose priorities are $[0, d]$.
- Principle: McNaughton-Zielonka’s algorithm with fixed tree of recursive calls.
Universal algorithm

- Goal: solve a parity game whose priorities are $[0, d]$.
- Principle: McNaughton-Zielonka’s algorithm with fixed tree of recursive calls.
Universal algorithm

- Goal: solve a parity game whose priorities are $[0, d]$.
- Principle: McNaughton-Zielonka’s algorithm with fixed tree of recursive calls.
- Inputs: $G$: game, $d$: top priority,
Universal algorithm

- Goal: solve a parity game whose priorities are $[0, d]$.
- Principle: McNaughton-Zielonka’s algorithm with fixed tree of recursive calls.
- Inputs: $G$: game, $d$: top priority, $T$: tree of height $d$. 
Universal algorithm

- Goal: solve a parity game whose priorities are $[0, d]$.
- Principle: McNaughton-Zielonka’s algorithm with fixed tree of recursive calls.
- Inputs: $G$: game, $d$: top priority, $T$: tree of height $d$.

1. $U$ = vertices from which Odd can win while avoiding $d$.
2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}^G_{\text{Odd}}(U)$.
3. Iterate: compute the Odd’s winning vertices in $G \setminus \text{Attr}^G_{\text{Odd}}(U)$. If you don’t find any, stop.
Universal algorithm

- Goal: solve a parity game whose priorities are $[0, d]$.
- Principle: McNaughton-Zielonka’s algorithm with fixed tree of recursive calls.
- Inputs: $G$: game, $d$: top priority, $T$: tree of height $d$.

1. $U = \text{vertices from which Odd can win while avoiding } d$.
2. Consider the set of vertices from which Odd can force the play to reach $U$, denoted $\text{Attr}_{\text{Odd}}^G(U)$.
3. Iterate: compute the Odd’s winning vertices in $G \times \text{Attr}_{\text{Odd}}^G(U)$. If you don’t find any, stop.
   Iterate $k$ times, where $k$ is the number of children of the root of $T$. 

R. Morvan
Fact: McNaughton-Zielonka’s algorithm corresponds to the universal algorithm over $(n, d)$-complete tree.
Fact: McNaughton-Zielonka’s algorithm corresponds to the universal algorithm over \((n, d)\)-complete tree.

\[ C_{n,d} = n \text{ copies of } C_{n,d-1} \]
Fact: Parys ('19) and Lehtinen-Schewe-Wojtczak ('19) algorithms are instances of the universal algorithm.
Fact: Parys (’19) and Lehtinen-Schewe-Wojtczak (’19) algorithms are instances of the universal algorithm.
Fact: Parys (’19) and Lehtinen-Schewe-Wojtczak (’19) algorithms are instances of the universal algorithm.
Fact: Parys ('19) and Lehtinen-Schewe-Wojtczak ('19) algorithms are instances of the universal algorithm.

$n$: number of vertices & $d$: top priority
Universal algorithm: correctness & complexity

- Time complexity of the universal algorithm over $(G, d, T)$: polynomial in $G$ and $T$. 
Universal algorithm: correctness & complexity

- Time complexity of the universal algorithm over \((G, d, T)\): polynomial in \(G\) and \(T\).
- Correctness: If \(T\) is big enough, then the algorithm is correct.
• Ordered trees: partially ordered by the “embedding” relation.
Ordered trees

- Ordered trees: partially ordered by the “embedding” relation.
- Correctness: For every game $G$, there exists a “small” tree $T_G$ such that the universal algorithm is correct whenever $T_G$ embeds in $T$. 
An attractor decomposition $Win_{\text{Even}}(G)$ of $G$ is a partition of the winning set describing the structure of Even’s winning set.

[Daviaud-Jurdziński-Lehtinen, ’18]
An attractor decomposition $\text{Win}_{\text{Even}}(G)$ of $G$ is a partition of the winning set describing the structure of Even’s winning set.

[Daviaud-Jurdziński-Lehtinen, ’18]
An attractor decomposition $\text{Win}_{\text{Even}}(G)$ of $G$ is a partition of the winning set describing the structure of Even’s winning set. [Daviaud-Jurdziński-Lehtinen, ’18]
An attractor decomposition $\text{Win}_{\text{Even}}(\mathcal{G})$ of $\mathcal{G}$ is a partition of the winning set describing the structure of Even’s winning set.

[Daviaud-Jurdziński-Lehtinen, ’18]
An attractor decomposition $\text{Win}_{\text{Even}}(\mathcal{G})$ of $\mathcal{G}$ is a partition of the winning set describing the structure of Even’s winning set. [Daviaud-Jurdziński-Lehtinen, ’18]
An attractor decomposition $\text{Win}_{\text{Even}}(\mathcal{G})$ of $\mathcal{G}$ is a partition of the winning set describing the structure of Even’s winning set.

[Daviaud-Jurdziński-Lehtinen, ’18]
An attractor decomposition $\text{Win}^{\text{Even}}(G)$ of $G$ is a partition of the winning set describing the structure of Even’s winning set.

[Daviaud-Jurdziński-Lehtinen, ’18]
Attractor decomposition (bis)
Attractor decomposition (ter)
Attractor decomposition (ter)
Attractor decomposition (ter)
Embeddable decomposition theorem

**Theorem:** If $D$ is subset of the winning set $W$ for Even, if Odd can force the play to stay in $D$, for every attractor decomposition tree $\mathcal{T}_W$ of $W$, there exists an attractor decomposition tree $\mathcal{T}_D$ of $D$ such that: $\mathcal{T}_D$ embeds in $\mathcal{T}_W$. 
**Embeddable decomposition theorem**

**Theorem:** If $D$ is subset of the winning set $W$ for Even, if Odd can force the play to stay in $D$, for every attractor decomposition tree $\mathcal{T}_W$ of $W$, there exists an attractor decomposition tree $\mathcal{T}_D$ of $D$ such that: $\mathcal{T}_D$ embeds in $\mathcal{T}_W$.

Attractor decomposition trees describe the shape of the structure of a winning region.
Correctness

- For every game $G$, there exists a “small” tree $T_G$ such that the universal algorithm is correct whenever $T_G$ embeds in $T$. 

R. Morvan
Correctness

- For every game $G$, there exists a “small” tree $T_G$ such that the universal algorithm is correct whenever $T_G$ embeds in $T$.
- Take $T_G = \text{product of an attractor decomposition tree for Even and an attractor decomposition tree for Odd.}$
Correctness

- For every game $G$, there exists a “small” tree $T_G$ such that the universal algorithm is correct whenever $T_G$ embeds in $T$.
- Take $T_G = \text{product of an attractor decomposition tree for Even and an attractor decomposition tree for Odd.}$
- Not unique.
Correctness

- For every game $G$, there exists a “small” tree $T_G$ such that the universal algorithm is correct whenever $T_G$ embeds in $T$.
- Take $T_G = \text{product of an attractor decomposition tree for Even and an attractor decomposition tree for Odd}$.
- Not unique.
- Polynomial size!
Universal trees

- A tree is \((n, d)\)-universal iff every tree with at most \(n\) leaves and of height \(d\) embeds in it.
Universal trees

- A tree is \((n, d)\)-universal iff every tree with at most \(n\) leaves and of height \(d\) embeds in it.
- Example: \((n, d)\)-complete tree.
Universal trees

- A tree is $(n, d)$-universal iff every tree with at most $n$ leaves and of height $d$ embeds in it.

- Example: $(n, d)$-complete tree.
Universal trees

- A tree is \((n, d)\)-universal iff every tree with at most \(n\) leaves and of height \(d\) embeds in it.
- Example: \((n, d)\)-complete tree.
- Example: \((n, d)\)-succinct tree.
Universal trees

- A tree is $(n, d)$-universal iff every tree with at most $n$ leaves and of height $d$ embeds in it.
- Example: $(n, d)$-complete tree.
- Example: $(n, d)$-succinct tree.
Universal trees & correctness

• For every game $G$, there exists a “small” tree $T_G$ such that the universal algorithm is correct whenever $T_G$ embeds in $T$. 
Universal trees & correctness

- For every game $G$, there exists a “small” tree $T_G$ such that the universal algorithm is correct whenever $T_G$ embeds in $T$.
- Works if $T_G$ is the product of two universal trees.
Universal trees & correctness

- For every game $G$, there exists a “small” tree $T_G$ such that the universal algorithm is correct whenever $T_G$ embeds in $T$.
- Works if $T_G$ is the product of two universal trees.
- This applies to McNaughton-Zielonka ’98, to Parys ’19 and to Lehtinen-Schewe-Wojtczak ’19.
Conclusion