Universal algorithms for parity games and nested fixpoints

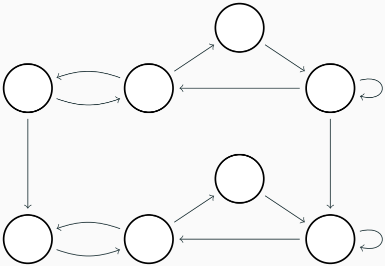
ANR DELTA meeting

Marcin Jurdziński[1], Rémi Morvan[2], K. S. Thejaswini[1]

June 28, 2021, in Paris!

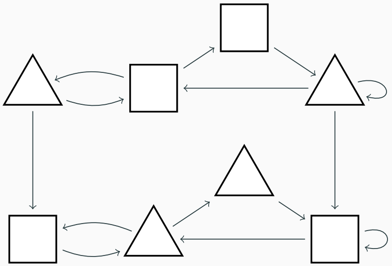
[1]University of Warwick
[2]École normale supérieure Paris-Saclay

Parity games



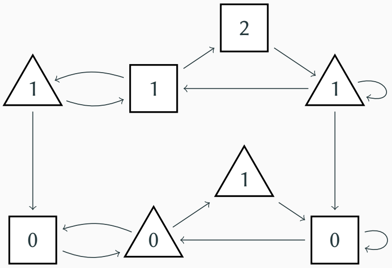
$$\mathcal{G} = \langle V, E, V_{\text{Even}}, V_{\text{Odd}}, \ \pi : V \longrightarrow [\![0, d]\!] \rangle$$

Parity games



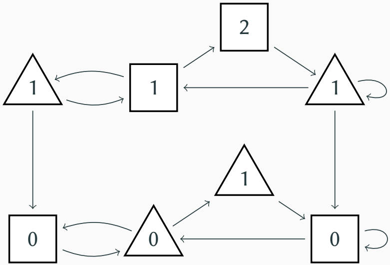
$$\mathcal{G} = \langle V, E, V_{\text{Even}}, V_{\text{Odd}},\ \pi : V \to [\![0, d]\!] \rangle$$

Parity games



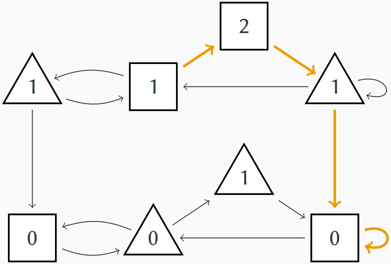
$$\mathcal{G} = \langle V, E, V_{\text{Even}}, V_{\text{Odd}},\ \pi : V \longrightarrow [\![0, d]\!] \rangle$$

Plays



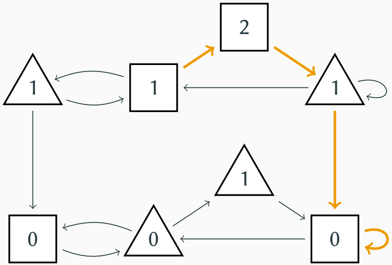
- Play: $(v_i)_{i \in \mathbb{N}}$ s.t. $\forall i,\ (v_i, v_{i+1}) \in E$.

Plays



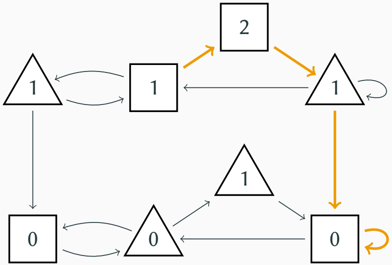
- Play: $(v_i)_{i \in \mathbb{N}}$ s.t. $\forall i,\ (v_i, v_{i+1}) \in E$.

Plays



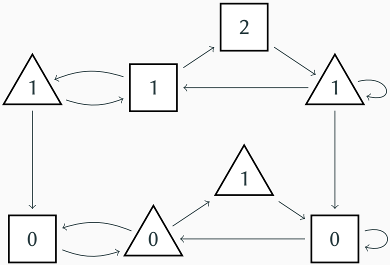
- Play: $(v_i)_{i \in \mathbb{N}}$ s.t. $\forall i,\ (v_i, v_{i+1}) \in E$.
- Even wins $(v_i)_{i \in \mathbb{N}}$ iff the maximal priority occuring infinitely often is even.

Plays



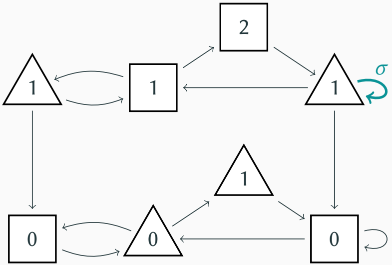
- Play: $(v_i)_{i \in \mathbb{N}}$ s.t. $\forall i,\ (v_i, v_{i+1}) \in E$.
- Even wins $(v_i)_{i \in \mathbb{N}}$ iff the maximal priority occuring infinitely often is even.

Strategies



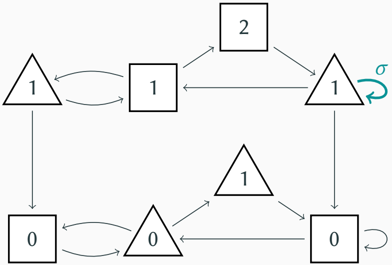
- Memoryless strategy for Odd: $\sigma : V_{\text{Odd}} \rightharpoonup V$ s.t. $(v, \sigma(v)) \in E$ for every vertex v.

Strategies



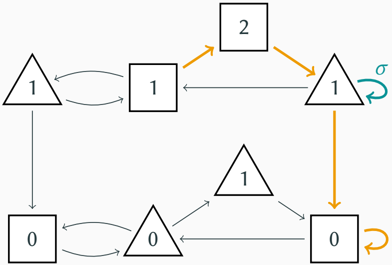
- Memoryless strategy for Odd: $\sigma : V_{\text{Odd}} \rightharpoonup V$ s.t. $(v, \sigma(v)) \in E$ for every vertex v.

Strategies



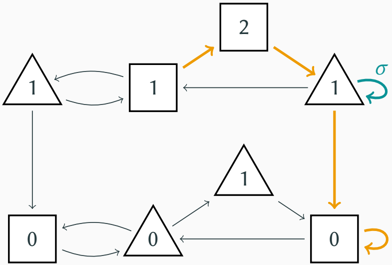
- Memoryless strategy for Odd: $\sigma : V_{\mathrm{Odd}} \rightharpoonup V$ s.t. $(v, \sigma(v)) \in E$ for every vertex v.
- A play $(v_i)_{i\in\mathbb{N}}$ is consistent with σ if $v_{i+1} = \sigma(v_i)$ whenever $\sigma(v_i)$ is defined.

Strategies



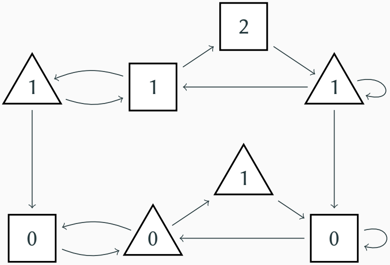
- Memoryless strategy for Odd: $\sigma : V_{\text{Odd}} \rightharpoonup V$ s.t. $(v, \sigma(v)) \in E$ for every vertex v.
- A play $(v_i)_{i\in\mathbb{N}}$ is consistent with σ if $v_{i+1} = \sigma(v_i)$ whenever $\sigma(v_i)$ is defined.

Strategies



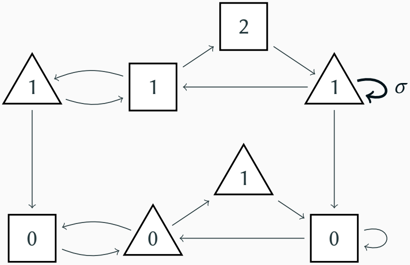
- Memoryless strategy for Odd: $\sigma : V_{\mathrm{Odd}} \rightharpoonup V$ s.t. $(v, \sigma(v)) \in E$ for every vertex v.
- A play $(v_i)_{i \in \mathbb{N}}$ is consistent with σ if $v_{i+1} = \sigma(v_i)$ whenever $\sigma(v_i)$ is defined.

Memoryless determinacy



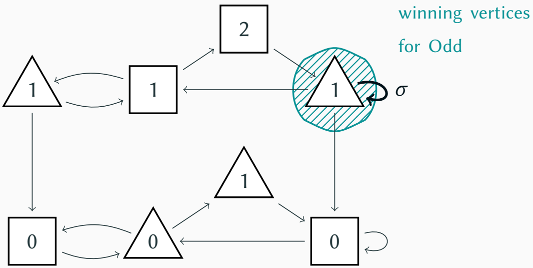
Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy σ such that every play consistent with σ and starting at v_0 is winning for her.

Memoryless determinacy



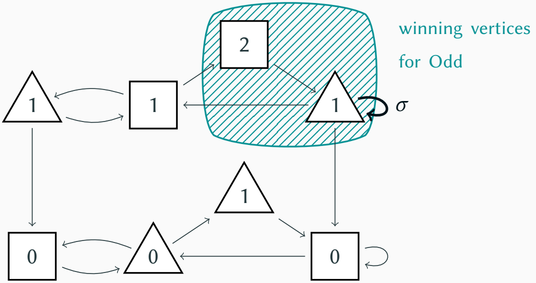
Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy σ such that every play consistent with σ and starting at v_0 is winning for her.

Memoryless determinacy



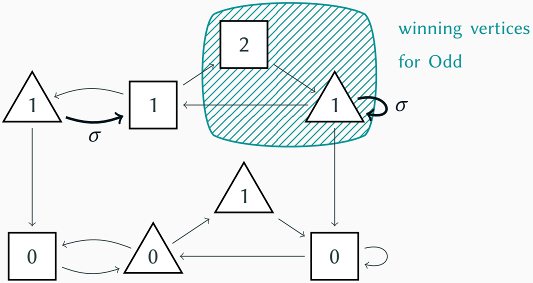
Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy σ such that every play consistent with σ and starting at v_0 is winning for her.

Memoryless determinacy



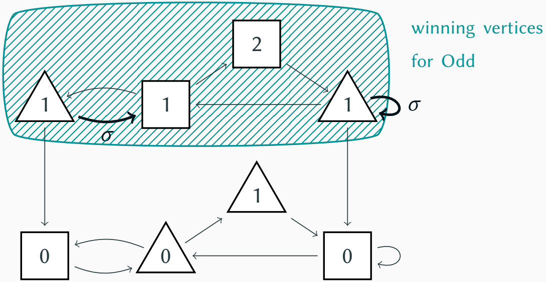
Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy σ such that every play consistent with σ and starting at v_0 is winning for her.

Memoryless determinacy



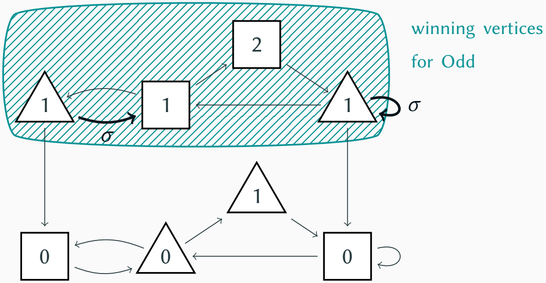
Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy σ such that every play consistent with σ and starting at v_0 is winning for her.

Memoryless determinacy



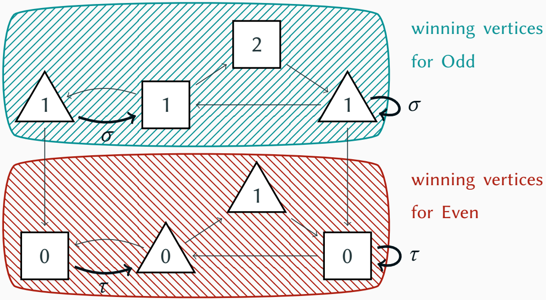
Theorem: From every vertex $v_0 \in V$, one of the two players has a memoryless strategy σ such that every play consistent with σ and starting at v_0 is winning for her.

Memoryless determinacy



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

Solving parity games

SOLVING PARITY GAMES:

Inputs: $\mathcal{G}$: parity game,

$v_0 \in V^{\mathcal{G}}$: vertex.

Question: Can player Even win from v_0 in $\mathcal{G}$?

Solving parity games

SOLVING PARITY GAMES:

Inputs: $\mathcal{G}$: parity game,
$v_0 \in V^{\mathcal{G}}$: vertex.

Question: Can player Even win from v_0 in $\mathcal{G}$?

- In NP ∩ coNP.

Solving parity games

SOLVING PARITY GAMES:

Inputs: $\mathcal{G}$: parity game,

$v_0 \in V^{\mathcal{G}}$: vertex.

Question: Can player Even win from v_0 in $\mathcal{G}$?

- In NP ∩ coNP.
- Believed to be in P…

Solving parity games

SOLVING PARITY GAMES:

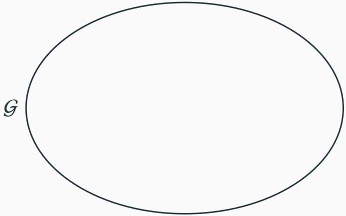
Inputs: $\mathcal{G}$: parity game,

$v_0 \in V^{\mathcal{G}}$: vertex.

Question: Can player Even win from v_0 in $\mathcal{G}$?

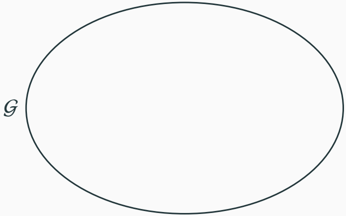
- In NP $\cap$ coNP.
- Believed to be in P…
- Best known upper bound: quasipolynomial time $O(n^{\log(d)})$ ['17 Calude-Jain-Khoussainov-Li-Stephan]

A recursive algorithm?



How to compute the winning vertices of Odd?

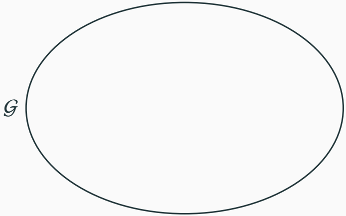
A recursive algorithm?



How to compute the winning vertices of Odd?

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

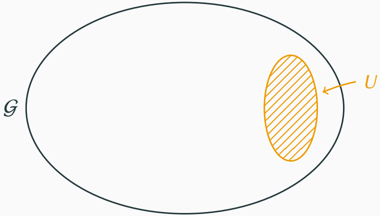
A recursive algorithm?



How to compute the winning vertices of Odd?

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

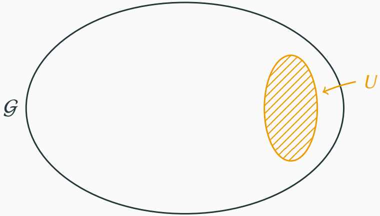
A recursive algorithm?



How to compute the winning vertices of Odd?

1. Identify a "small" winning set U for Odd. How???

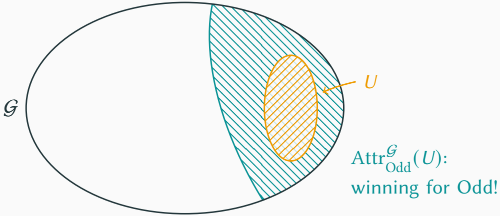
A recursive algorithm?



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.

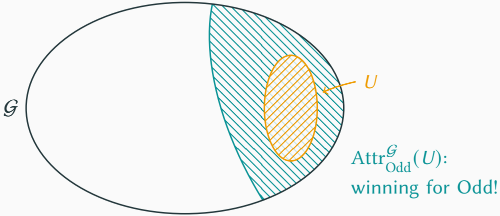
A recursive algorithm?



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.

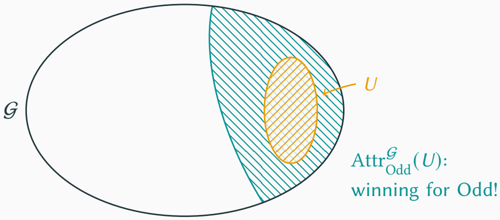
A recursive algorithm?



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd’s winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.

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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd’s winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. 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

$$\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Even}}(\pi^{-1}[d]).$$

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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.

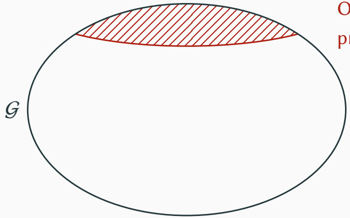
McNaughton-Zielonka's algorithm

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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.

McNaughton-Zielonka's algorithm

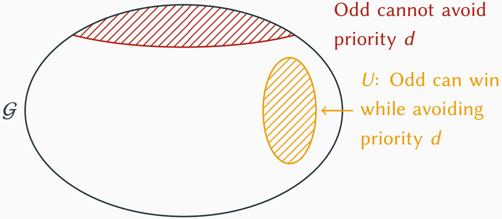
Recursive algorithm. At each call: fewer priorities or fewer vertices.



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.

McNaughton-Zielonka's algorithm

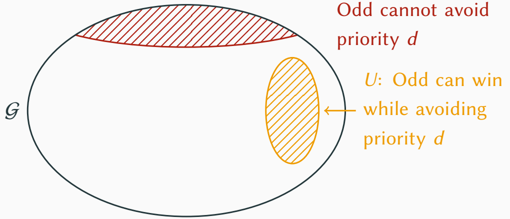
Recursive algorithm. At each call: fewer priorities or fewer vertices.



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.

McNaughton-Zielonka's algorithm

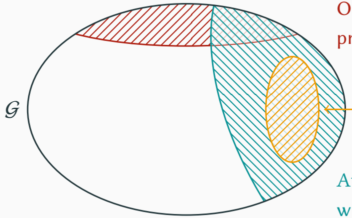
Recursive algorithm. At each call: fewer priorities or fewer vertices.



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.

McNaughton-Zielonka's algorithm

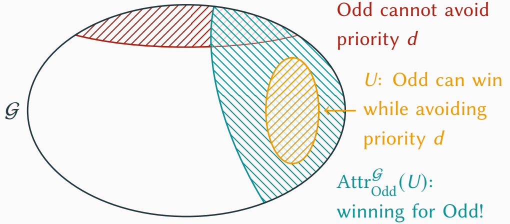
Recursive algorithm. At each call: fewer priorities or fewer vertices.



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 $\mathrm{Attr}_{\mathrm{Odd}}^{\mathcal{G}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}_{\mathrm{Odd}}^{\mathcal{G}}(U)$. If you don't find any, stop.

McNaughton-Zielonka's algorithm

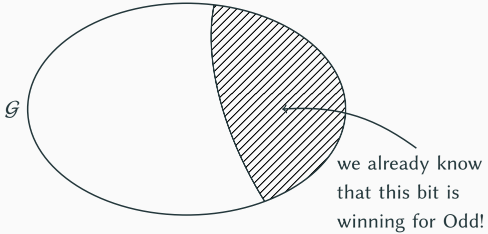
Recursive algorithm. At each call: fewer priorities or fewer vertices.



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.

McNaughton-Zielonka's algorithm

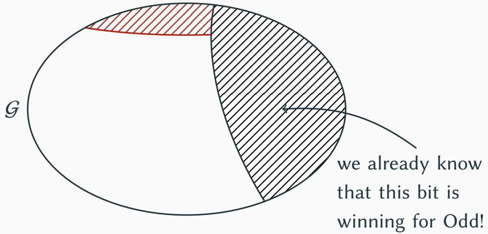
Recursive algorithm. At each call: fewer priorities or fewer vertices.



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.

McNaughton-Zielonka's algorithm

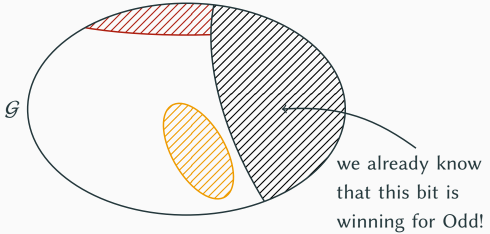
Recursive algorithm. At each call: fewer priorities or fewer vertices.



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.

McNaughton-Zielonka's algorithm

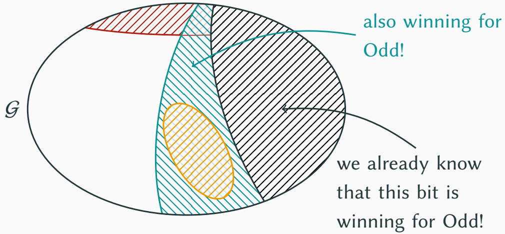
Recursive algorithm. At each call: fewer priorities or fewer vertices.



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.

McNaughton-Zielonka's algorithm

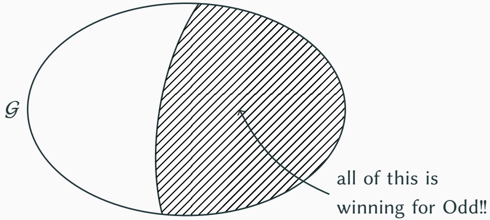
Recursive algorithm. At each call: fewer priorities or fewer vertices.



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. 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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.

McNaughton-Zielonka's algorithm: correctness

- Correctness: by construction.

McNaughton-Zielonka's algorithm: correctness

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

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 ← easy to identify!
  - at least one, but finitely many
  - ~~infinitely many~~"

McNaughton-Zielonka's algorithm: correctness

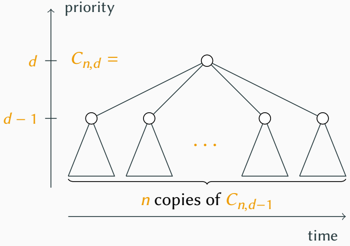
- Correctness: by construction.
- Completeness: ~~???~~ by construction!

  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 ← easy to identify!
  - at least one, but finitely many
  - ~~infinitely many~~"

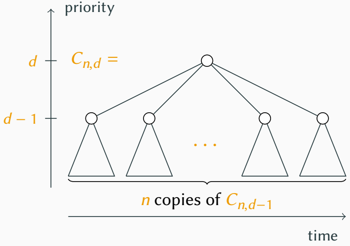
McNaughton-Zielonka's algorithm: complexity

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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute the Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.



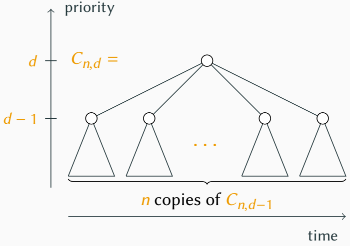
McNaughton-Zielonka's algorithm: complexity

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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute the Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.



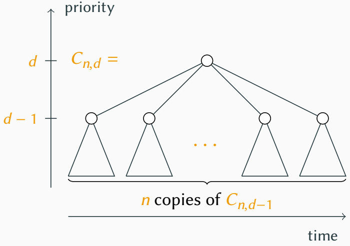
McNaughton-Zielonka's algorithm: complexity

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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. Computable in poly. time!
3. Iterate: compute the Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop.



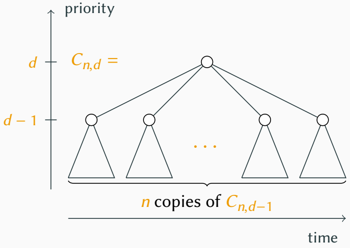
McNaughton-Zielonka's algorithm: complexity

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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. Computable in poly. time!
3. Iterate: compute the Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. 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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. Computable in poly. time!
3. Iterate: compute the Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$. If you don't find any, stop. How many times will we need to iterate? At most $n = |V|$.



Universal algorithm

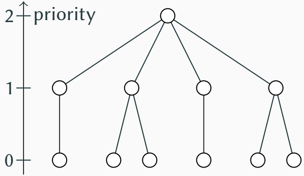
- Goal: solve a parity game whose priorities are $[\![0, d]\!]$.

Universal algorithm

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

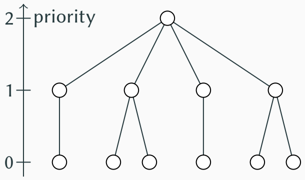
Universal algorithm

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



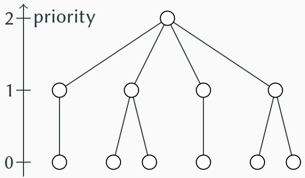
Universal algorithm

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



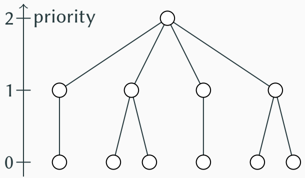
Universal algorithm

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



Universal algorithm

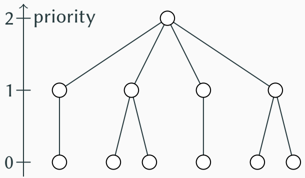
- Goal: solve a parity game whose priorities are $[\![0, d]\!]$.
- Principle: McNaughton-Zielonka's algorithm with fixed tree of recursive calls.
- Inputs: $\mathcal{G}$: game, d: top priority, $\mathcal{T}$: tree of height d.



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 $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. Iterate: compute the Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
   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: $\mathcal{G}$: game, d: top priority, $\mathcal{T}$: tree of height d.



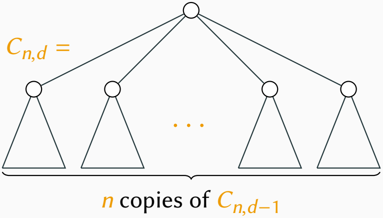
1. U = vertices from which Odd can win while avoiding d.
2. Consider the set of vertices from which Odd can force the play to reach U, denoted $\mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.
3. ~~Iterate: compute the Odd's winning vertices in $\mathcal{G} \setminus \mathrm{Attr}^{\mathcal{G}}_{\mathrm{Odd}}(U)$.~~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

McNaughton-Zielonka

Fact: McNaughton-Zielonka's algorithm corresponds to the universal algorithm over (n, d)-complete tree.

McNaughton-Zielonka

Fact: McNaughton-Zielonka's algorithm corresponds to the universal algorithm over (n, d)-complete tree.

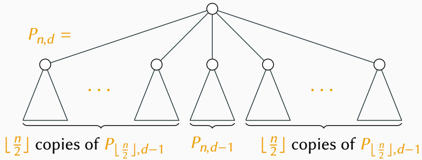


Parys & Lehtinen-Schewe-Wojtczak

Fact: Parys ('19) and Lehtinen-Schewe-Wojtczak ('19) algorithms are instances of the universal algorithm.

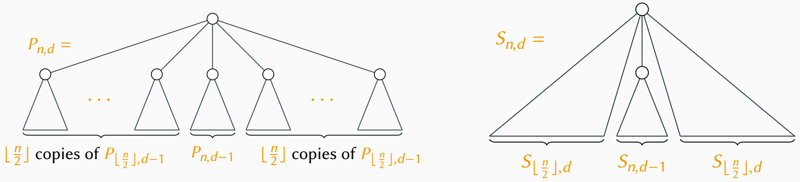
Parys & Lehtinen-Schewe-Wojtczak

Fact: Parys ('19) and Lehtinen-Schewe-Wojtczak ('19) algorithms are instances of the universal algorithm.



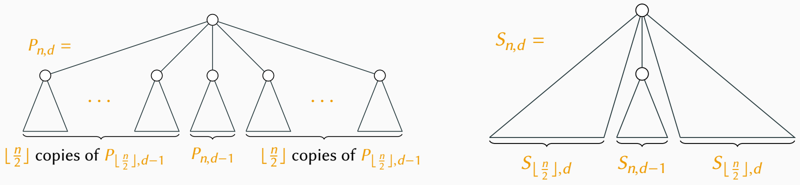
Parys & Lehtinen-Schewe-Wojtczak

Fact: Parys ('19) and Lehtinen-Schewe-Wojtczak ('19) algorithms are instances of the universal algorithm.



Parys & Lehtinen-Schewe-Wojtczak

Fact: Parys ('19) and Lehtinen-Schewe-Wojtczak ('19) algorithms are instances of the universal algorithm.



n: number of vertices & d: top priority

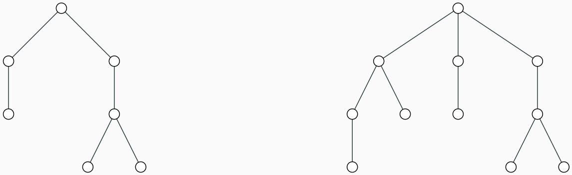
Universal algorithm: correctness & complexity

- Time complexity of the universal algorithm over $(\mathcal{G}, d, \mathcal{T})$: polynomial in $\mathcal{G}$ and $\mathcal{T}$.

Universal algorithm: correctness & complexity

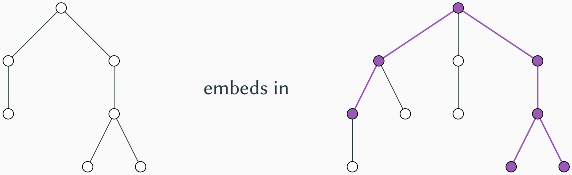
- Time complexity of the universal algorithm over $(\mathcal{G}, d, \mathcal{T})$: polynomial in $\mathcal{G}$ and $\mathcal{T}$.
- Correctness: If $\mathcal{T}$ is big enough, then the algorithm is correct.

Ordered trees



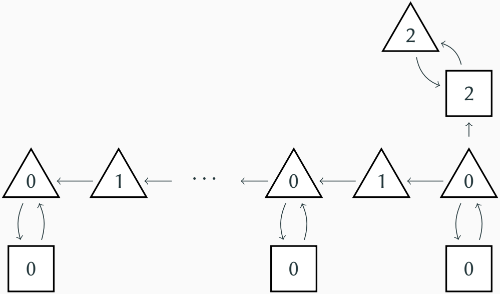
- Ordered trees: partially ordered by the “embedding” relation.

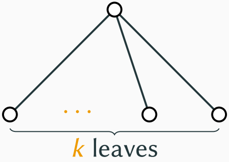
Ordered trees



- Ordered trees: partially ordered by the "embedding" relation.
- Correctness: For every game $\mathcal{G}$, there exists a "small" tree $\mathcal{T}_\mathcal{G}$ such that the universal algorithm is correct whenever $\mathcal{T}_\mathcal{G}$ embeds in $\mathcal{T}$.

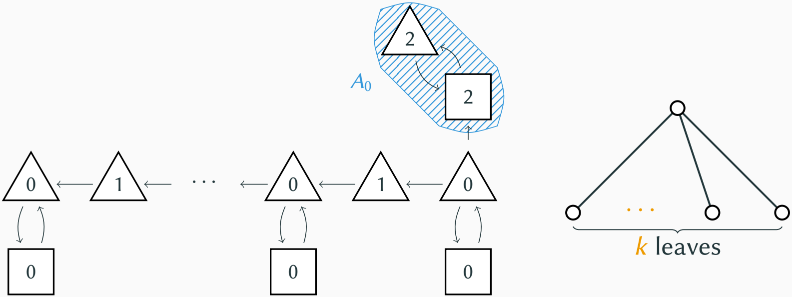
Attractor decomposition





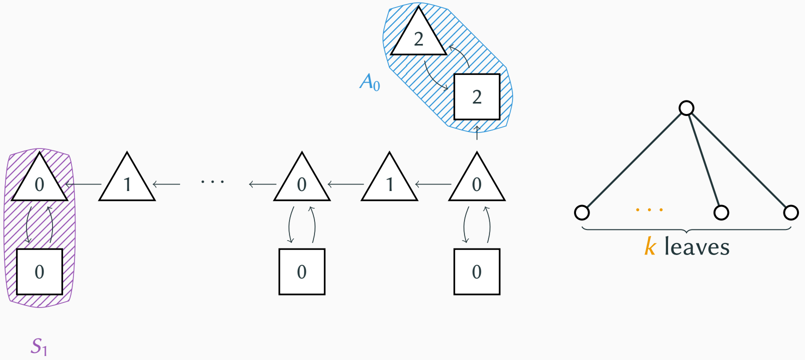
An attractor decomposition $\mathrm{Win}_{\mathrm{Even}}(\mathcal{G})$ of $\mathcal{G}$ is a partition of the winning set describing the structure of Even's winning set. [Daviaud-Jurdziński-Lehtinen, '18]

Attractor decomposition



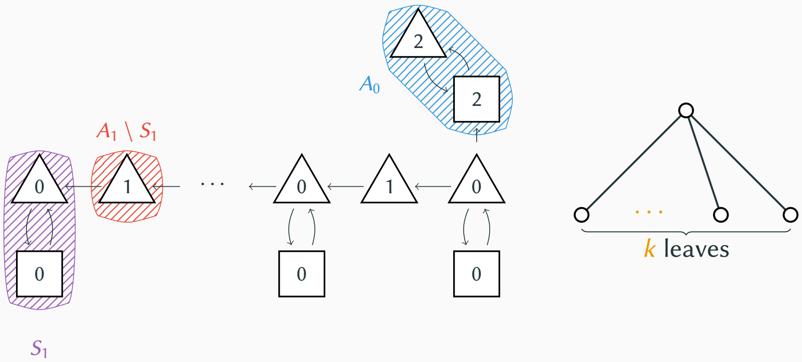
An attractor decomposition $\mathrm{Win}_{\mathrm{Even}}(\mathcal{G})$ of $\mathcal{G}$ is a partition of the winning set describing the structure of Even's winning set. [Daviaud-Jurdziński-Lehtinen, '18]

Attractor decomposition



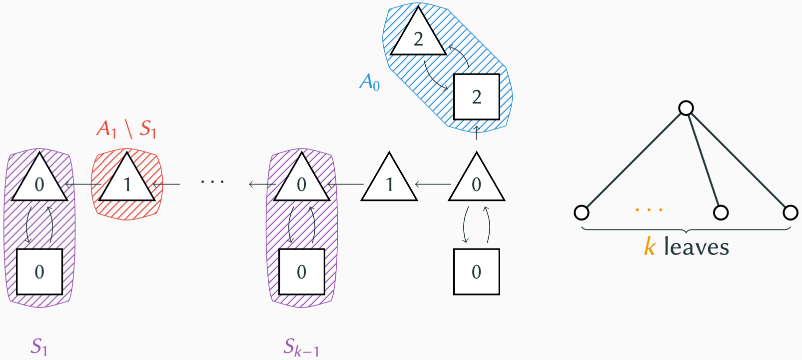
An attractor decomposition $\mathrm{Win}_{\mathrm{Even}}(\mathcal{G})$ of $\mathcal{G}$ is a partition of the winning set describing the structure of Even's winning set. [Daviaud-Jurdziński-Lehtinen, '18]

Attractor decomposition



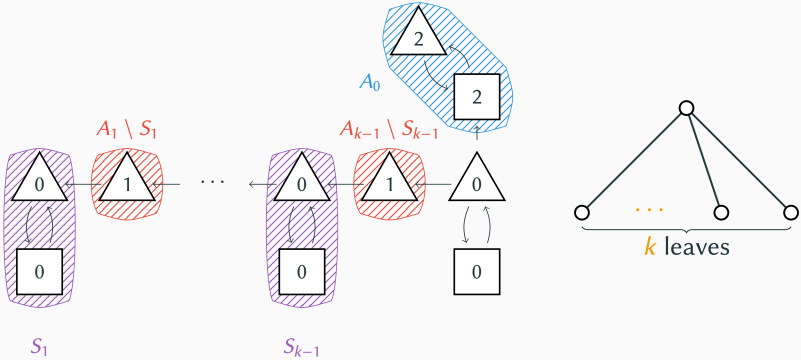
An attractor decomposition $\mathrm{Win}_{\mathrm{Even}}(\mathcal{G})$ of $\mathcal{G}$ is a partition of the winning set describing the structure of Even's winning set. [Daviaud-Jurdziński-Lehtinen, '18]

Attractor decomposition



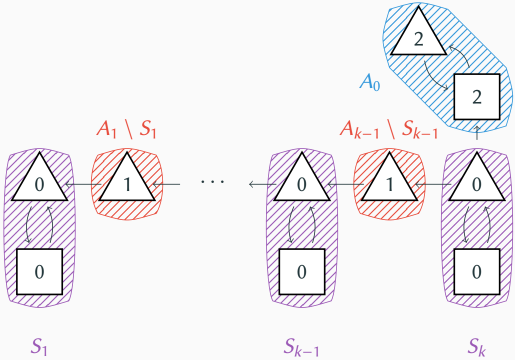
An attractor decomposition $\mathrm{Win}_{\mathrm{Even}}(\mathcal{G})$ of $\mathcal{G}$ is a partition of the winning set describing the structure of Even's winning set. [Daviaud-Jurdziński-Lehtinen, '18]

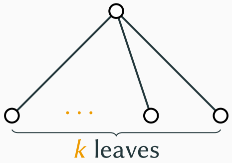
Attractor decomposition



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

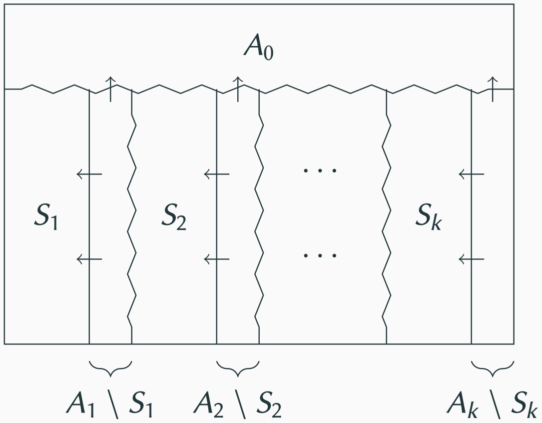
Attractor decomposition



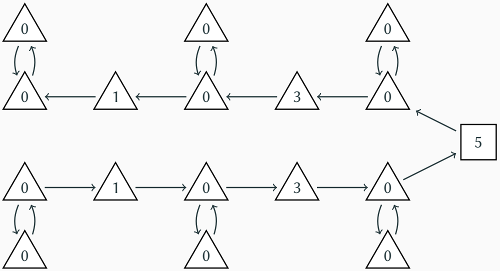


An attractor decomposition $\mathrm{Win}_{\mathrm{Even}}(\mathcal{G})$ of $\mathcal{G}$ is a partition of the winning set describing the structure of Even's winning set. [Daviaud-Jurdziński-Lehtinen, '18]

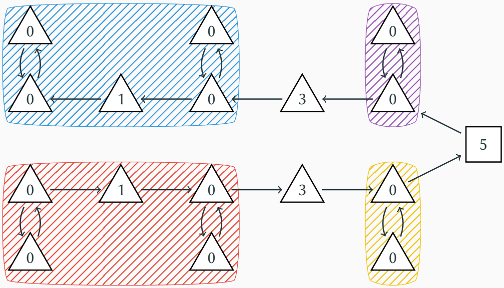
Attractor decomposition (bis)



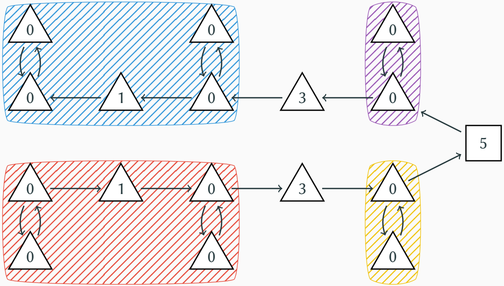
Attractor decomposition (ter)

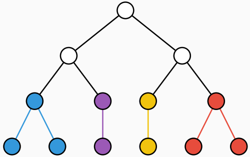


Attractor decomposition (ter)



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 $\mathcal{G}$, there exists a “small” tree $\mathcal{T}_{\mathcal{G}}$ such that the universal algorithm is correct whenever $\mathcal{T}_{\mathcal{G}}$ embeds in $\mathcal{T}$.

Correctness

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

Correctness

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

Correctness

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

Universal trees

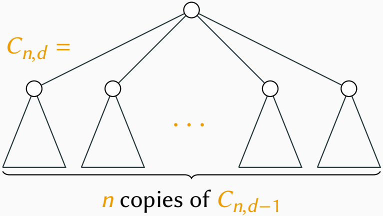
- A tree is (n, d)-universal iff every tree with at most n leaves and of height d embeds in it.

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.

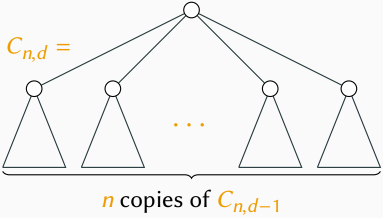
Universal trees

- 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.



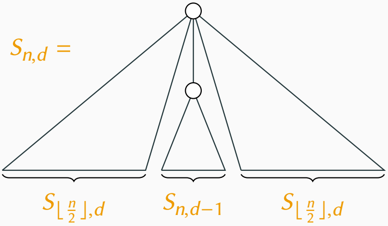
Universal trees

- 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 $\mathcal{G}$, there exists a "small" tree $\mathcal{T}_{\mathcal{G}}$ such that the universal algorithm is correct whenever $\mathcal{T}_{\mathcal{G}}$ embeds in $\mathcal{T}$.

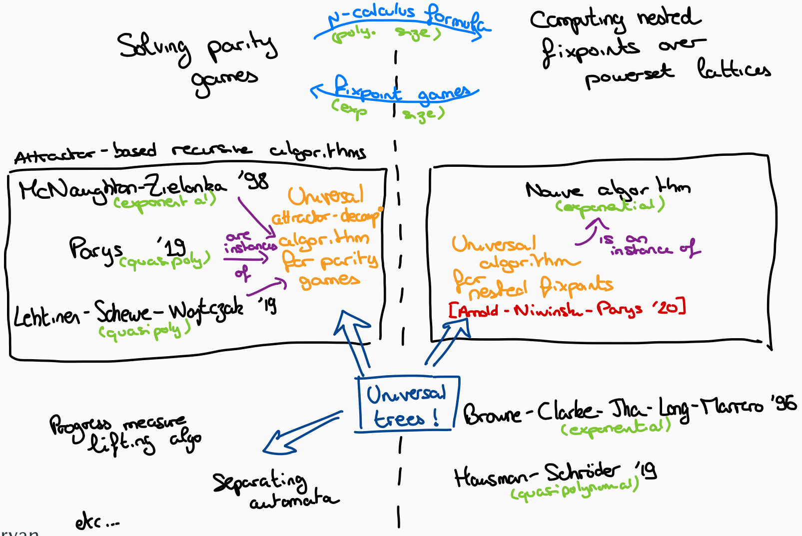
Universal trees & correctness

- For every game $\mathcal{G}$, there exists a “small” tree $\mathcal{T}_{\mathcal{G}}$ such that the universal algorithm is correct whenever $\mathcal{T}_{\mathcal{G}}$ embeds in $\mathcal{T}$.
- Works if $\mathcal{T}$ is the product of two universal trees.

Universal trees & correctness

- For every game $\mathcal{G}$, there exists a "small" tree $\mathcal{T}_{\mathcal{G}}$ such that the universal algorithm is correct whenever $\mathcal{T}_{\mathcal{G}}$ embeds in $\mathcal{T}$.
- Works if $\mathcal{T}$ is the product of two universal trees.
- This applies to McNaughton-Zielonka '98, to Parys '19 and to Lehtinen-Schewe- Wojtczak '19.

Conclusion


Solving parity games
μ-calculus formula
(poly. size)
Fixpoint games
(exp size)
Computing nested fixpoints over powerset lattices
Attractor-based recursive algorithms
McNaughton-Zielonka '98
(exponential)
Parys '19
(quasipoly)
Lehtinen-Schewe-Wojtczak '19
(quasipoly)
are instances of
Universal attractor-decomp algorithm for parity games
Naive algorithm
(exponential)
is an instance of
Universal algorithm for nested fixpoints
[Arnold-Niwinski-Parys '20]
Universal trees!
Progress measure lifting algo
Separating automata
etc...
Browne-Clarke-Jha-Long-Marrero '96
(exponential)
Hausman-Schröder '19
(quasipolynomial)