Reactive Synthesis

Lecture 9

Swen Jacobs and Martin Zimmermann
(Saarland University)

Plan for Today

- Basic Games
- Algorithms & Data Structures
- Advanced Games

- Temporal Logic Synthesis

Plan for Today

- Basic Games
- Algorithms & Data Structures
- Advanced Games
  - Parity Games
- Temporal Logic Synthesis

Motivation

Four foundational winning conditions:

Reachability reaching a goal (at least once)

Safety staying safe (at all times)

Recurrence reaching a goal infinitely often

Persistence staying safe from some point onwards

Motivation

Four foundational winning conditions:

Reachability reaching a goal (at least once)

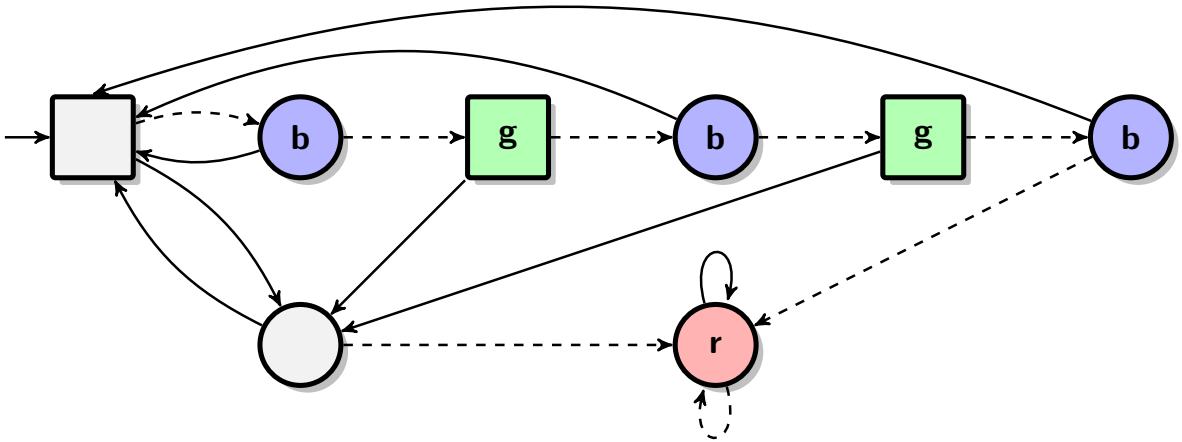
Safety staying safe (at all times)

Recurrence reaching a goal infinitely often

Persistence staying safe from some point onwards

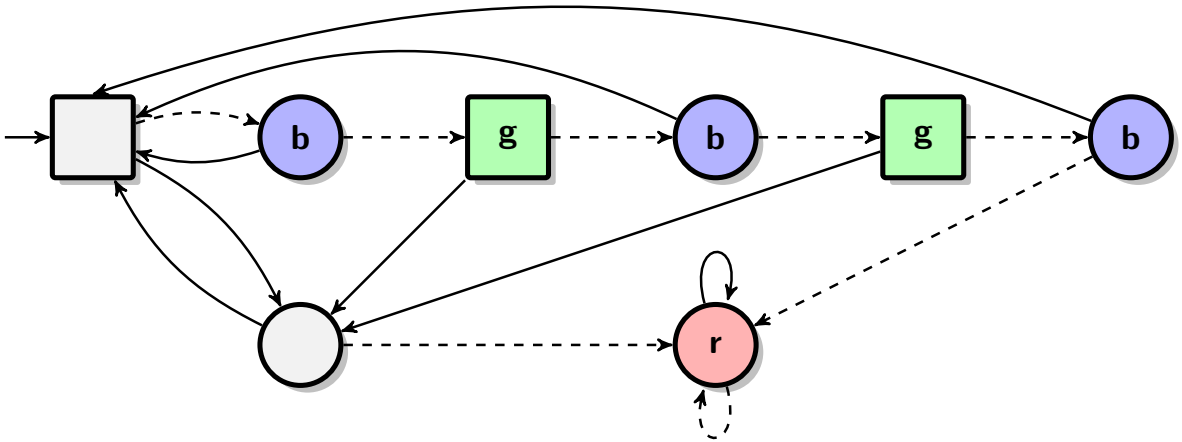
There are still specifications not expressible with these conditions, e.g., Boolean combinations of basic conditions.

Back in Lecture 1



- Never visit red vertex and
- if infinitely many blue vertices are visited, then infinitely many green vertices are visited.

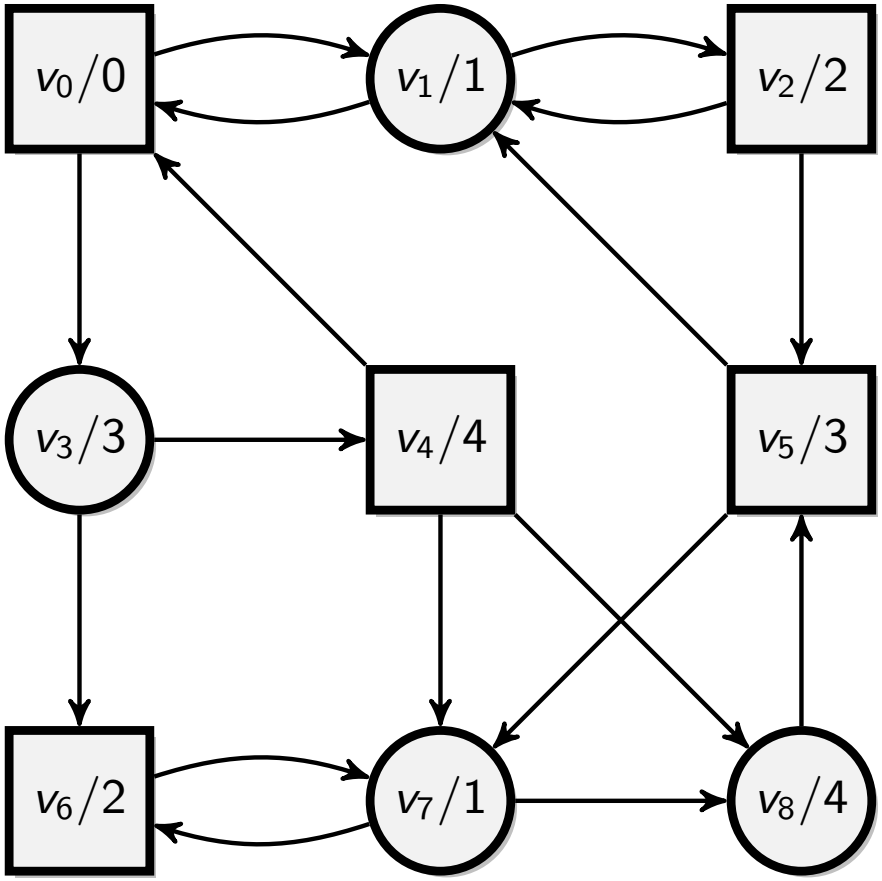
Back in Lecture 1



- Never visit red vertex and
- if infinitely many blue vertices are visited, then infinitely many green vertices are visited.

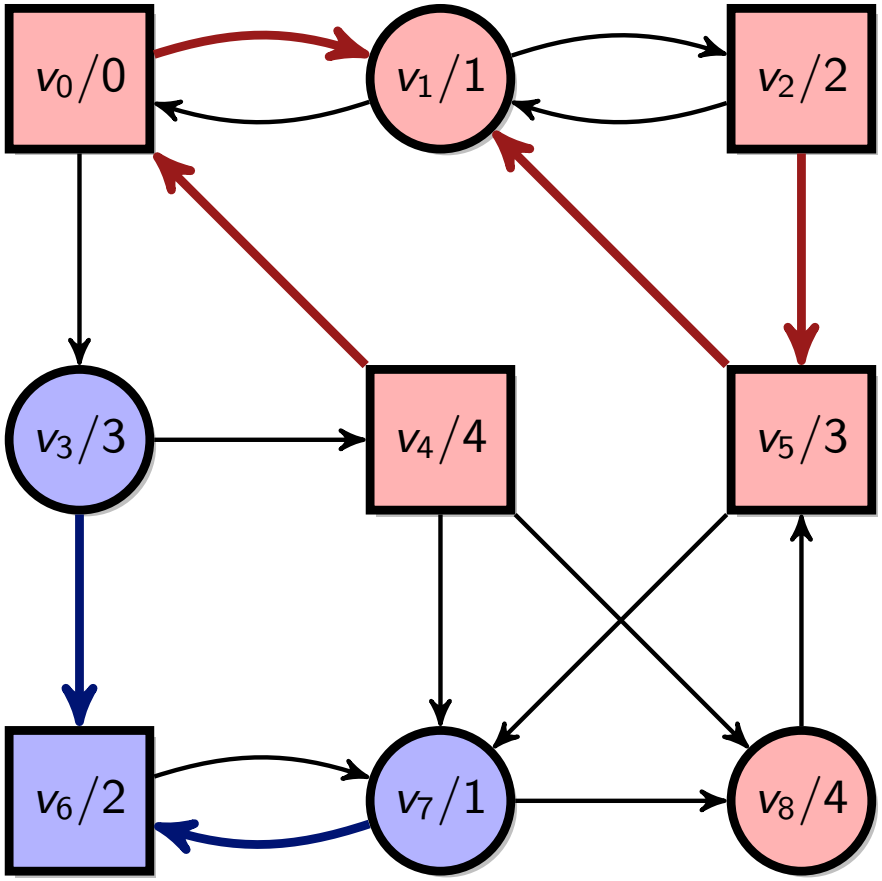
$$\textsc{Safety}(\textbf{not red}) \cap (\textsc{coBüchi}(\textbf{not blue}) \cup \textsc{Büchi}(\textbf{green}))$$

Parity Games



Player 0 wins if the maximal color visited infinitely often is even.

Parity Games



Player 0 wins if the maximal color visited infinitely often is even.

Parity Games

Recall

$\mathrm{Inf}(\rho) := \{v \in V \mid v_n = v \text{ for infinitely many } n\}$: the set of vertices occurring infinitely often in ρ.

Definition

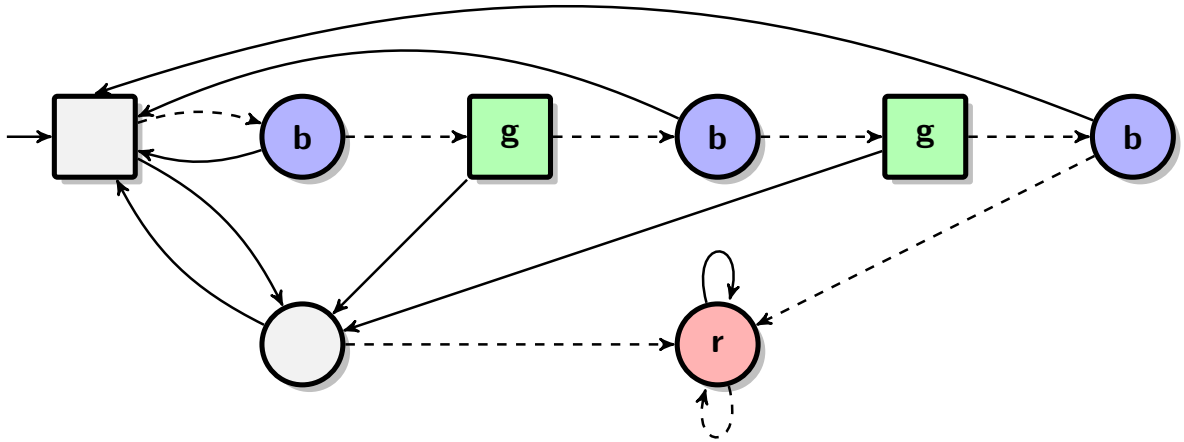
Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena and let $\Omega\colon V \to \mathbb{N}$ be a *coloring* of $\mathcal{A}$'s vertices. Then, the *parity condition* $\textsc{Parity}(\Omega)$ is defined as

$$\textsc{Parity}(\Omega) := \{\rho \in V^\omega \mid \max \mathrm{Inf}(\Omega(\rho_0)\Omega(\rho_1)\Omega(\rho_2)\cdots) \text{ is even}\}.$$

We call a game $\mathcal{G} = (\mathcal{A}, \textsc{Parity}(\Omega))$ a *parity game*.

Intuition: color c is desirable for Player c mod 2.

Back in Lecture 1



- Never visit red vertex and
- if infinitely many blue vertices are visited, then infinitely many green vertices are visited.

This can be expressed as a parity condition

- **gray** vertices $\mapsto 0$
- **blue** vertices $\mapsto 1$
- **red** vertices $\mapsto 3$
- **green** vertices $\mapsto 2$

Applications

Parity games play a central role in automata and logics:

- Normal form for ω-regular languages: deterministic parity automata (next week).
- Model-checking games of the modal μ-calculus.
- Emptiness of parity tree automata is equivalent to parity games.
- Semantics of alternating automata on infinite objects.
- Intriguing complexity-theoretic status (later today).

Properties

Lemma

Every parity condition is a Boolean combination of Büchi conditions.

Properties

Lemma

Every parity condition is a Boolean combination of Büchi conditions.

Lemma

1. *Büchi conditions are exactly those parity conditions with an odd color c_1 and an even color $c_2 > c_1$.*
2. *Co-Büchi conditions are exactly those parity conditions with an even color c_0 and an odd color $c_1 > c_0$.*

Properties

Lemma

Every parity condition is a Boolean combination of Büchi conditions.

Lemma

1. *Büchi conditions are exactly those parity conditions with an odd color c_1 and an even color $c_2 > c_1$.*
2. *Co-Büchi conditions are exactly those parity conditions with an even color c_0 and an odd color $c_1 > c_0$.*

Lemma

The parity condition is self-dual, i.e., $\mathrm{Par}(\Omega) = V^\omega \setminus \mathrm{Par}(\Omega')$ where $\Omega'(v) = \Omega(v) + 1$.

More Properties: Prefix-independence

Definition

A winning condition $\mathrm{Win} \subseteq V^\omega$ is prefix-independent, if adding or removing a prefix of a play does not change the winner, e.g., if $\rho \in \mathrm{Win} \Leftrightarrow w\rho \in \mathrm{Win}$ for all $\rho \in V^\omega$ and all $w \in V^*$.

More Properties: Prefix-independence

Definition

A winning condition $\mathrm{Win} \subseteq V^\omega$ is prefix-independent, if adding or removing a prefix of a play does not change the winner, e.g., if $\rho \in \mathrm{Win} \Leftrightarrow w\rho \in \mathrm{Win}$ for all $\rho \in V^\omega$ and all $w \in V^*$.

Remark

- *Parity conditions are prefix-independent.*
- *Reachability and safety conditions are not prefix-independent.*

More Properties: Prefix-independence

Definition

A winning condition $\mathrm{Win} \subseteq V^\omega$ is prefix-independent, if adding or removing a prefix of a play does not change the winner, e.g., if $\rho \in \mathrm{Win} \Leftrightarrow w\rho \in \mathrm{Win}$ for all $\rho \in V^\omega$ and all $w \in V^*$.

Remark

- *Parity conditions are prefix-independent.*
- *Reachability and safety conditions are not prefix-independent.*

Lemma

Let σ be a winning strategy for Player i from a set W of vertices in a game with prefix-independent winning condition.
If a play ρ has a suffix starting in W that is consistent with σ, then ρ is winning for Player i.

More Properties: Traps

Definition

Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena. A set $T \subseteq V$ is a *trap* for Player i, if

- every vertex $v \in T \cap V_i$ has only successors in T, i.e., $(v, v') \in E$ implies $v' \in T$, and
- every vertex $v \in T \cap V_{1-i}$ there is a successor in T, i.e., there is some $v' \in T$ with $(v, v') \in E$.

More Properties: Traps

Definition

Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena. A set $T \subseteq V$ is a *trap* for Player i, if

- every vertex $v \in T \cap V_i$ has only successors in T, i.e., $(v, v') \in E$ implies $v' \in T$, and
- every vertex $v \in T \cap V_{1-i}$ there is a successor in T, i.e., there is some $v' \in T$ with $(v, v') \in E$.

Remark

The complement of a Player i attractor is a trap for Player i.

More Properties: Traps

Definition

Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena. A set $T \subseteq V$ is a *trap* for Player i, if

- every vertex $v \in T \cap V_i$ has only successors in T, i.e., $(v, v') \in E$ implies $v' \in T$, and
- every vertex $v \in T \cap V_{1-i}$ there is a successor in T, i.e., there is some $v' \in T$ with $(v, v') \in E$.

Remark

The complement of a Player i attractor is a trap for Player i.

Lemma

Let $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ be a game with prefix-independent winning condition Win. Then, $W_i(\mathcal{G})$ is a trap for Player $1 - i$.

Main Theorem

Theorem

Parity games are determined with positional winning strategies.

Main Theorem

Theorem

Parity games are determined with positional winning strategies.

Proof.

By induction over the number n of vertices.

Main Theorem

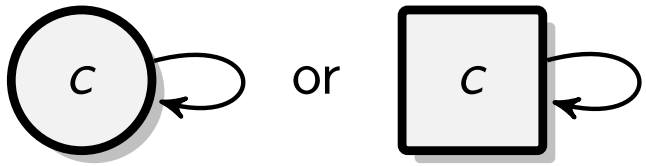
Theorem

Parity games are determined with positional winning strategies.

Proof.

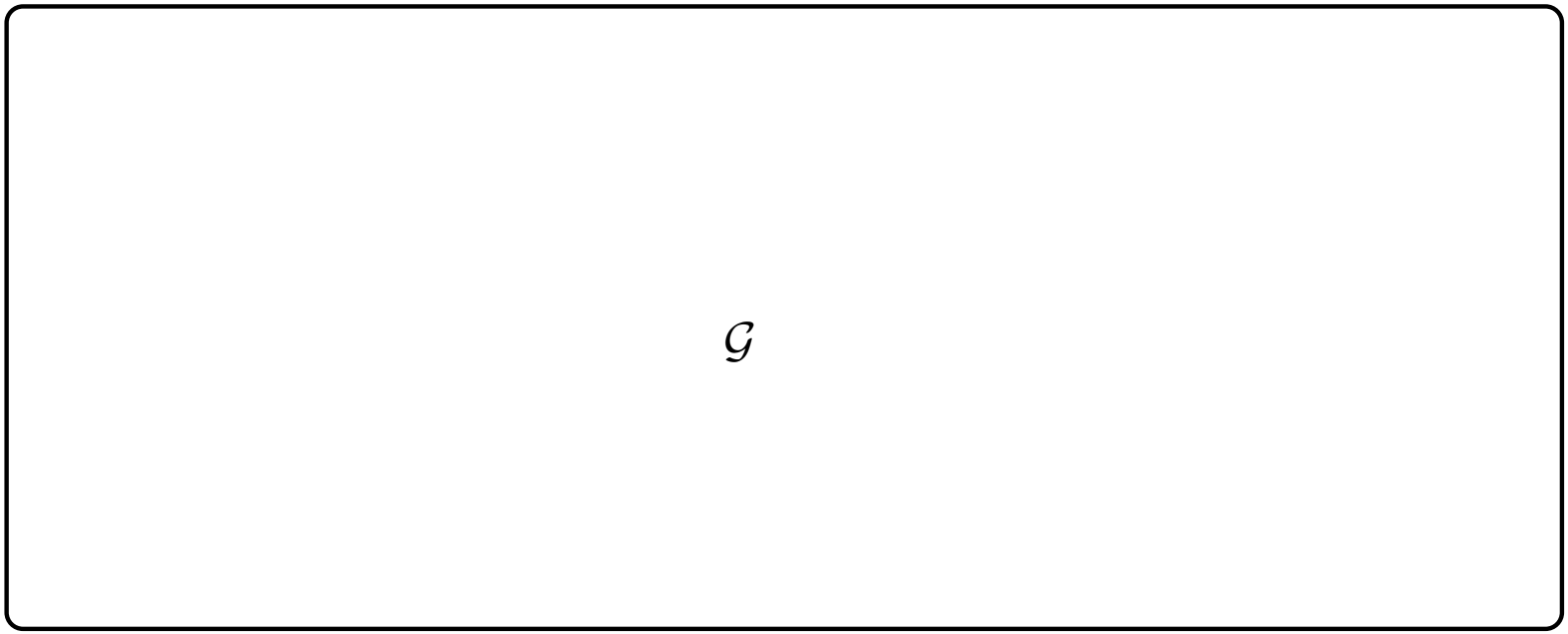
By induction over the number n of vertices.

- The induction start $n = 1$ is trivial:

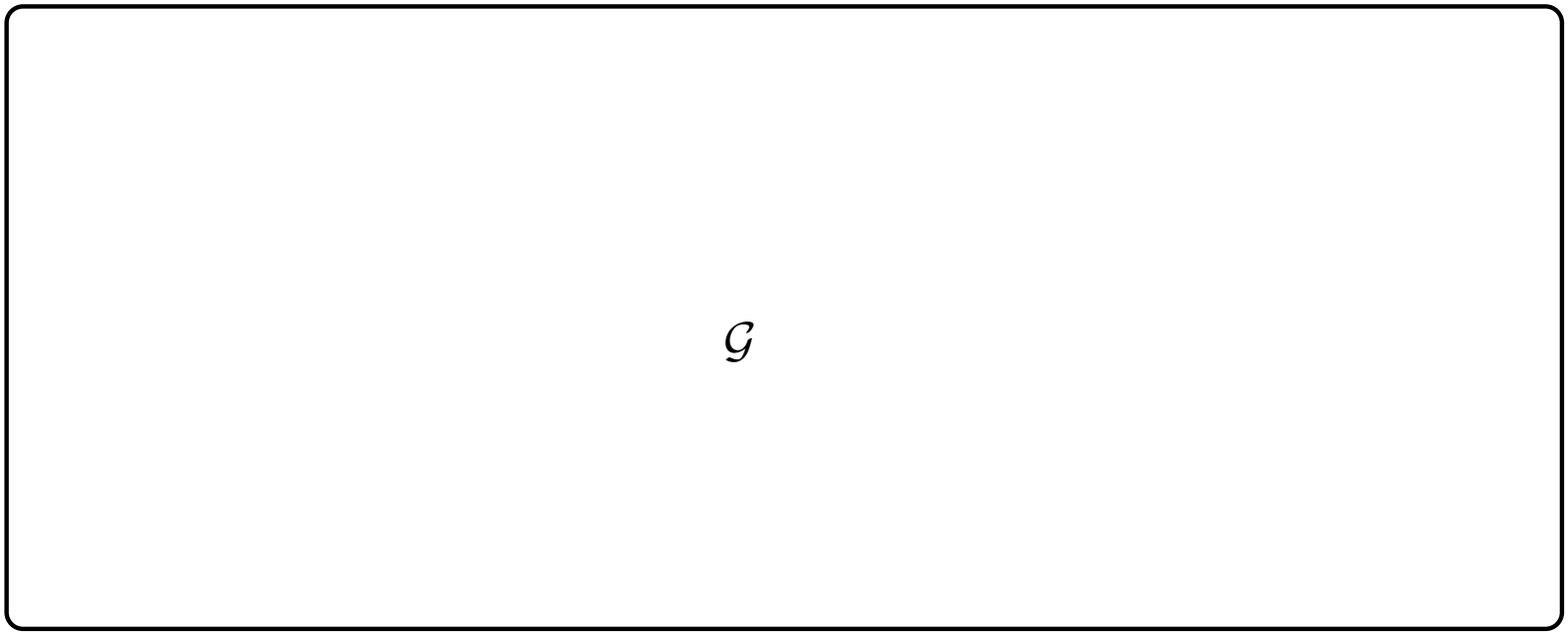


- Player (c mod 2) wins with a positional strategy.

Proof

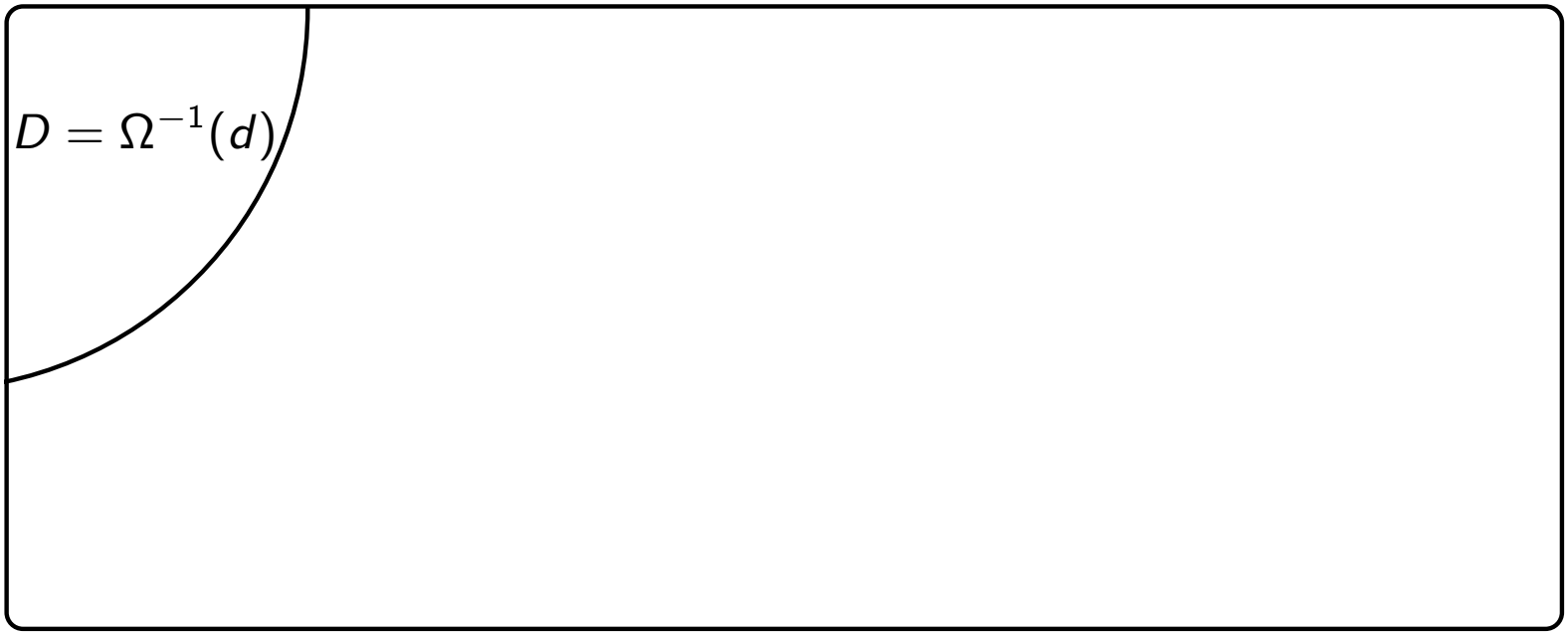


Proof



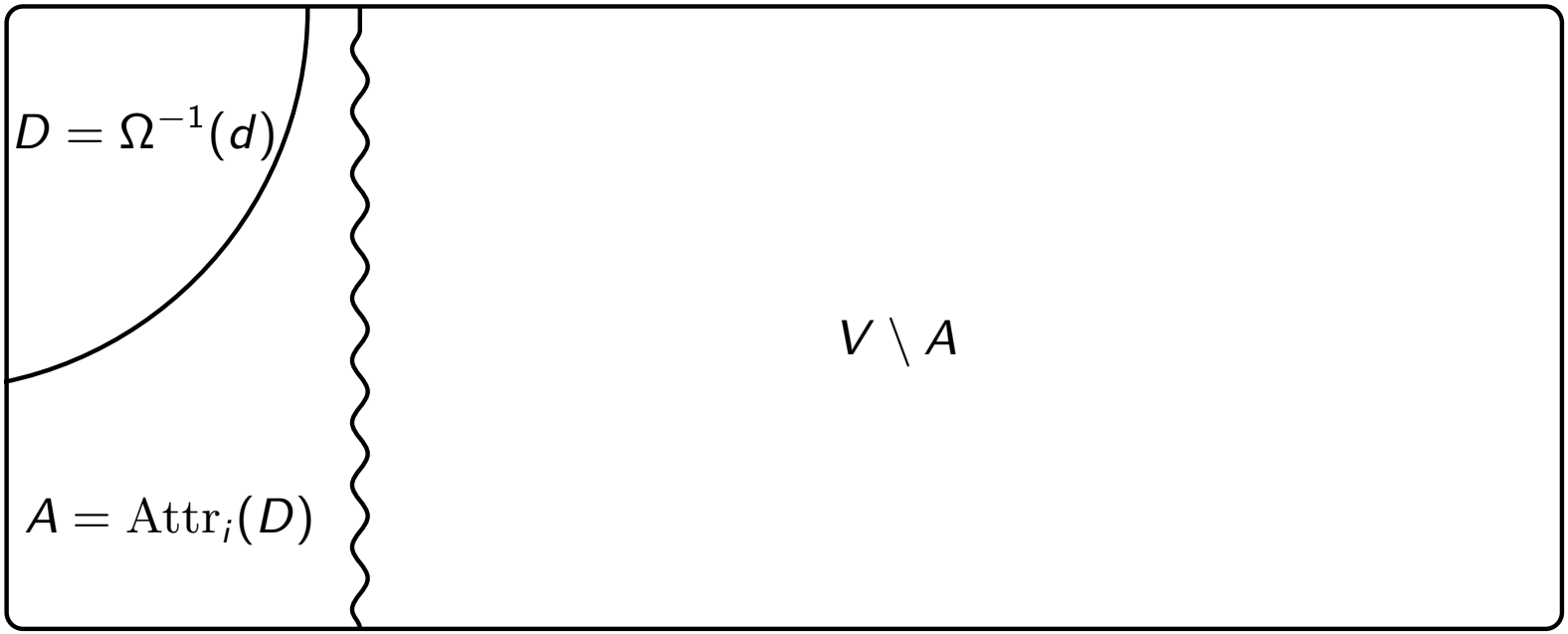
Let d be the maximal color in $\mathcal{G}$ and define $i = d \bmod 2$, i.e., d is desirable for Player i.

Proof



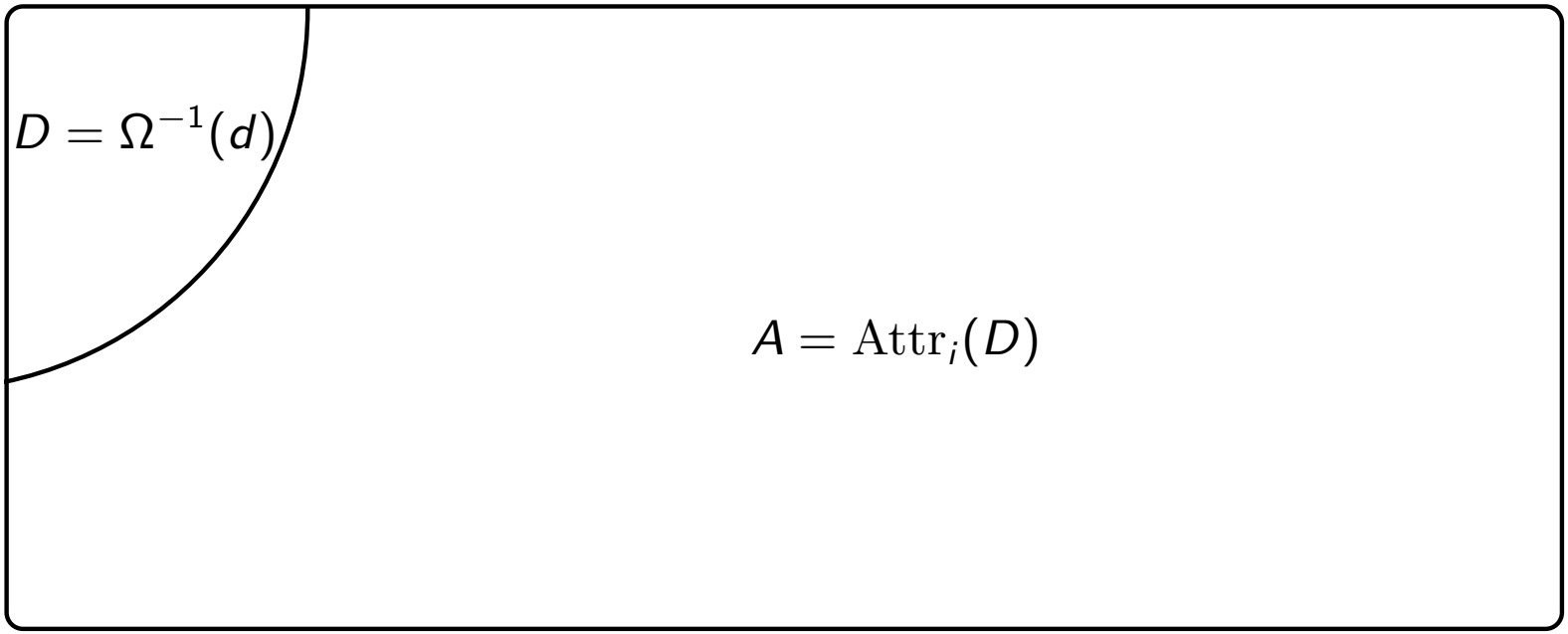
Let d be the maximal color in $\mathcal{G}$ and define $i = d \bmod 2$, i.e., d is desirable for Player i.

Proof



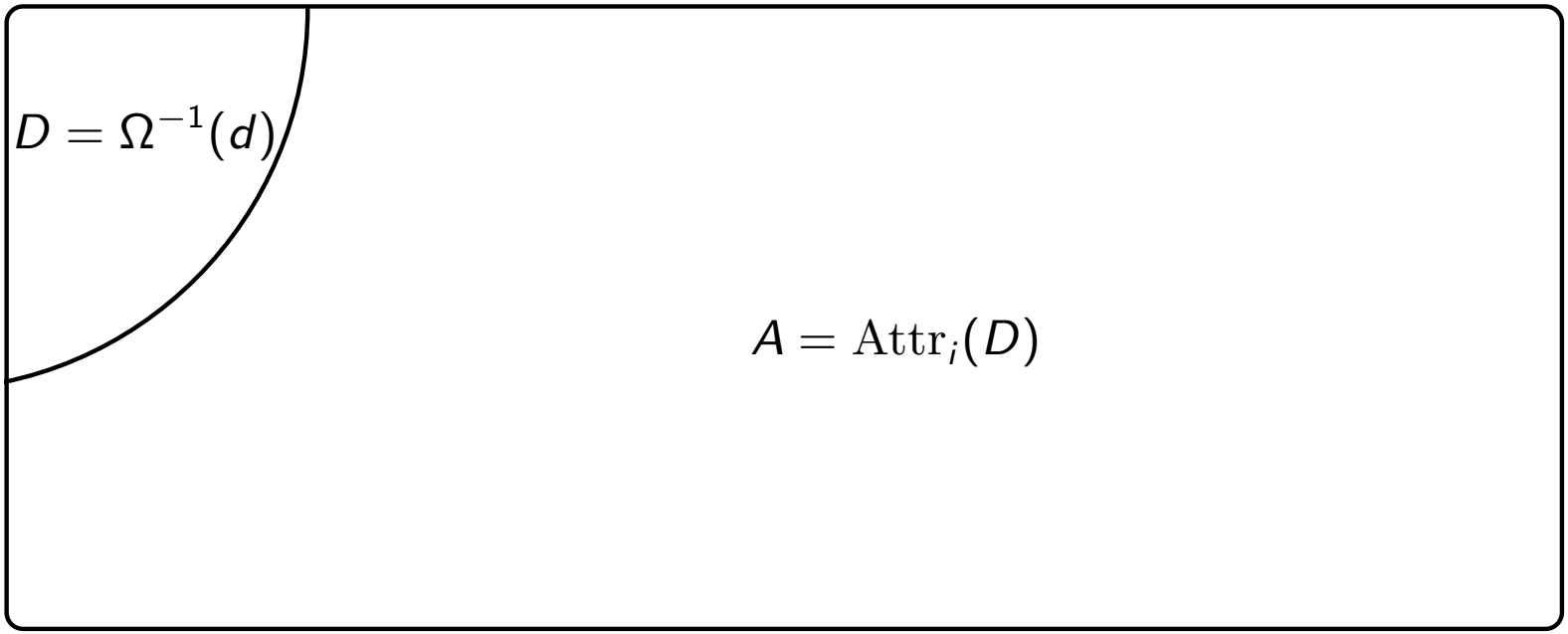
Let d be the maximal color in $\mathcal{G}$ and define $i = d \bmod 2$, i.e., d is desirable for Player i.

Proof



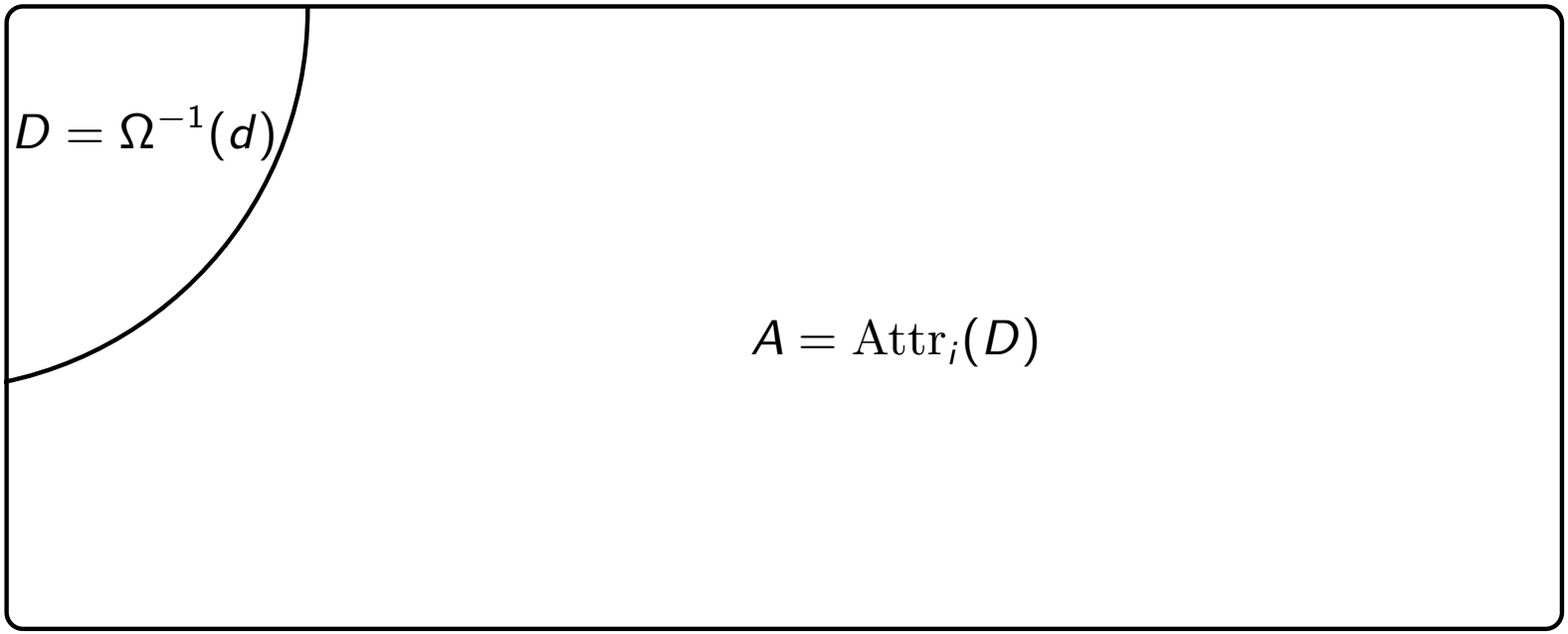
First assume $A = V$.

Proof



Claim $W_i(\mathcal{G}) = V$:

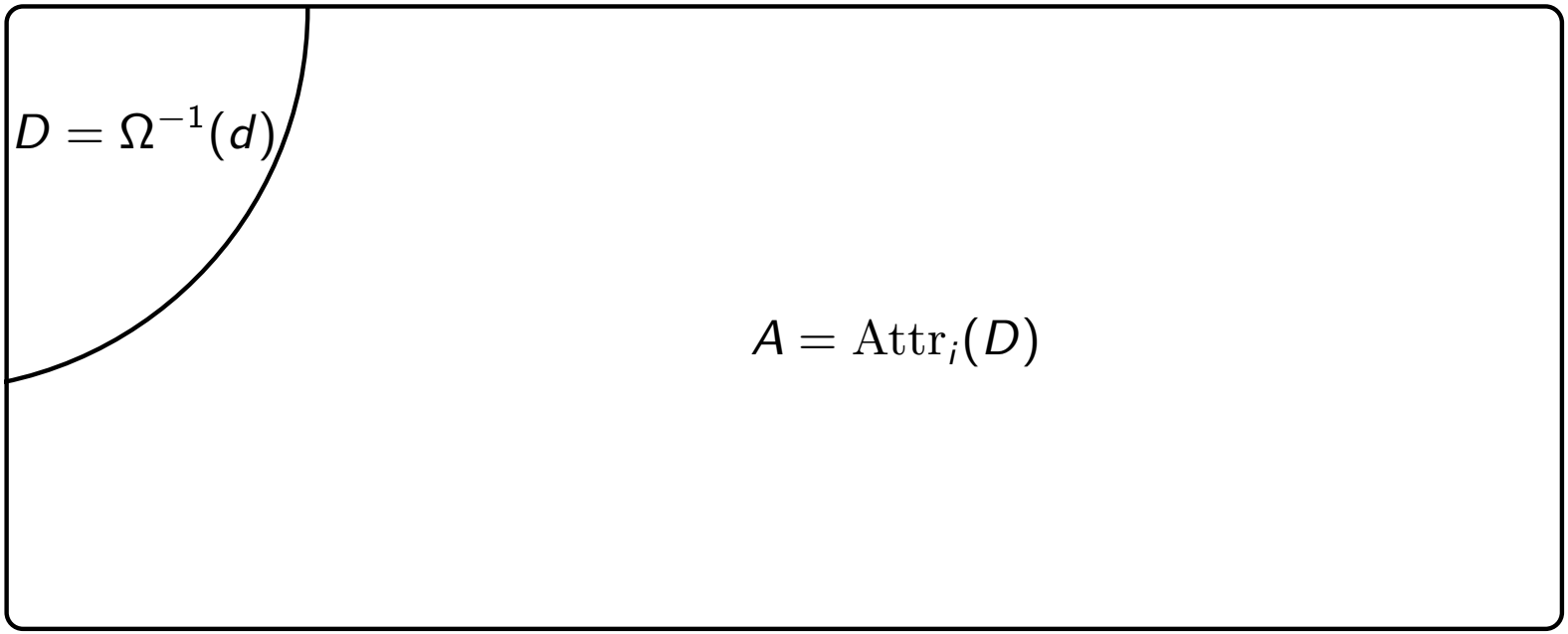
Proof



Claim $W_i(\mathcal{G}) = V$:

- In $A \setminus D$ use attractor strategy.
- In D move arbitrarily.

Proof

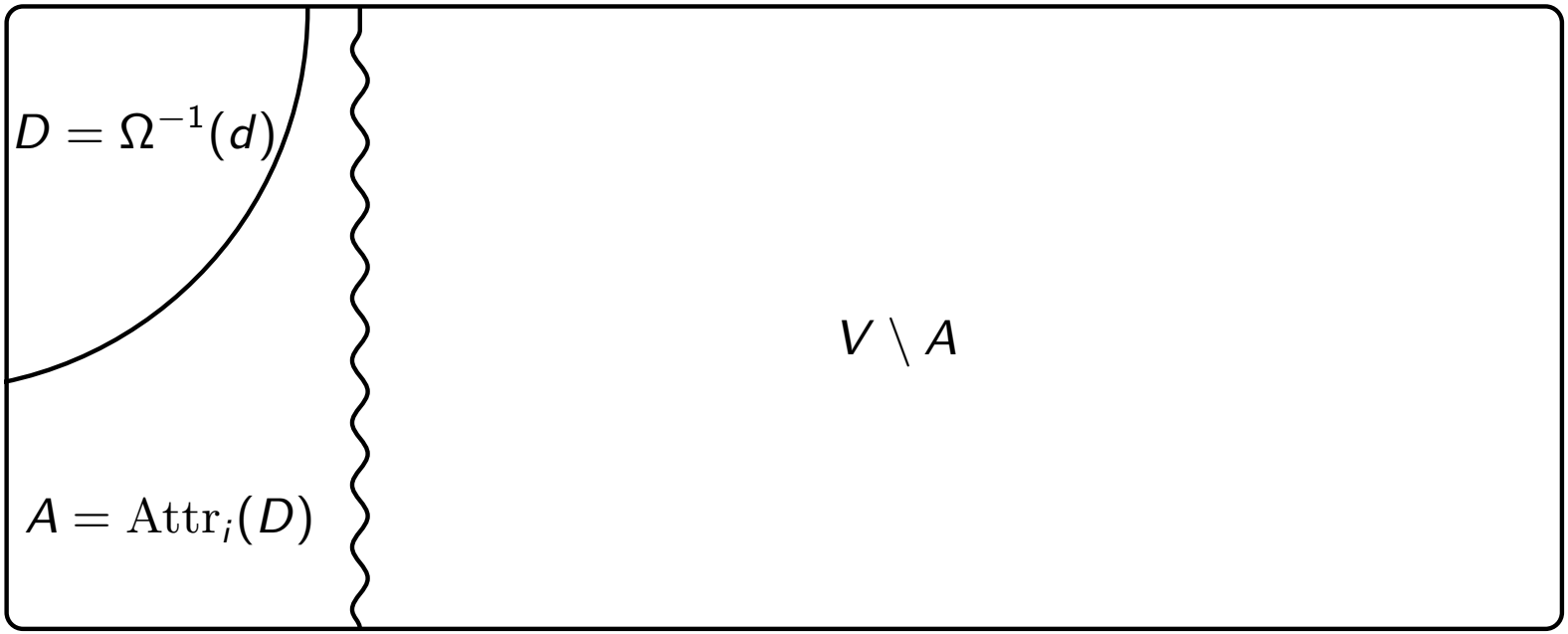


Claim $W_i(\mathcal{G}) = V$:

- In $A \setminus D$ use attractor strategy.
- In D move arbitrarily.

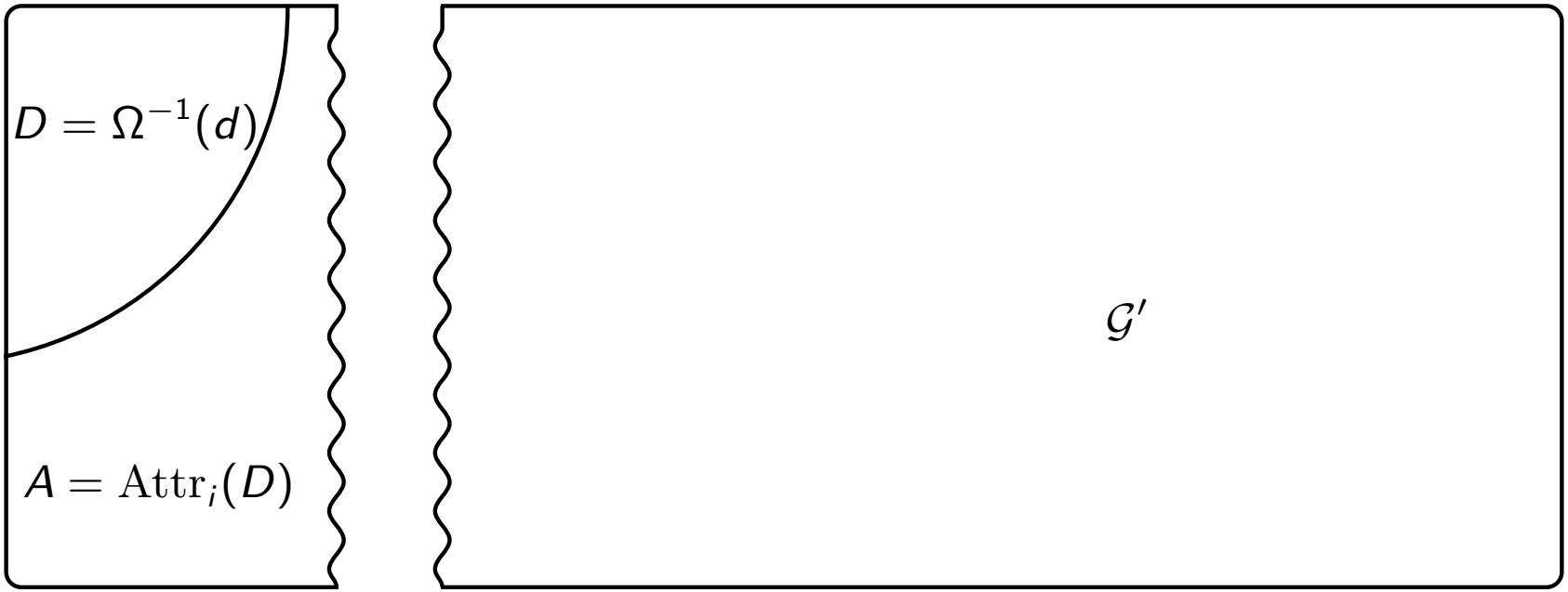
Every play consistent with this strategy visits D infinitely often and is therefore winning for Player i.

Proof

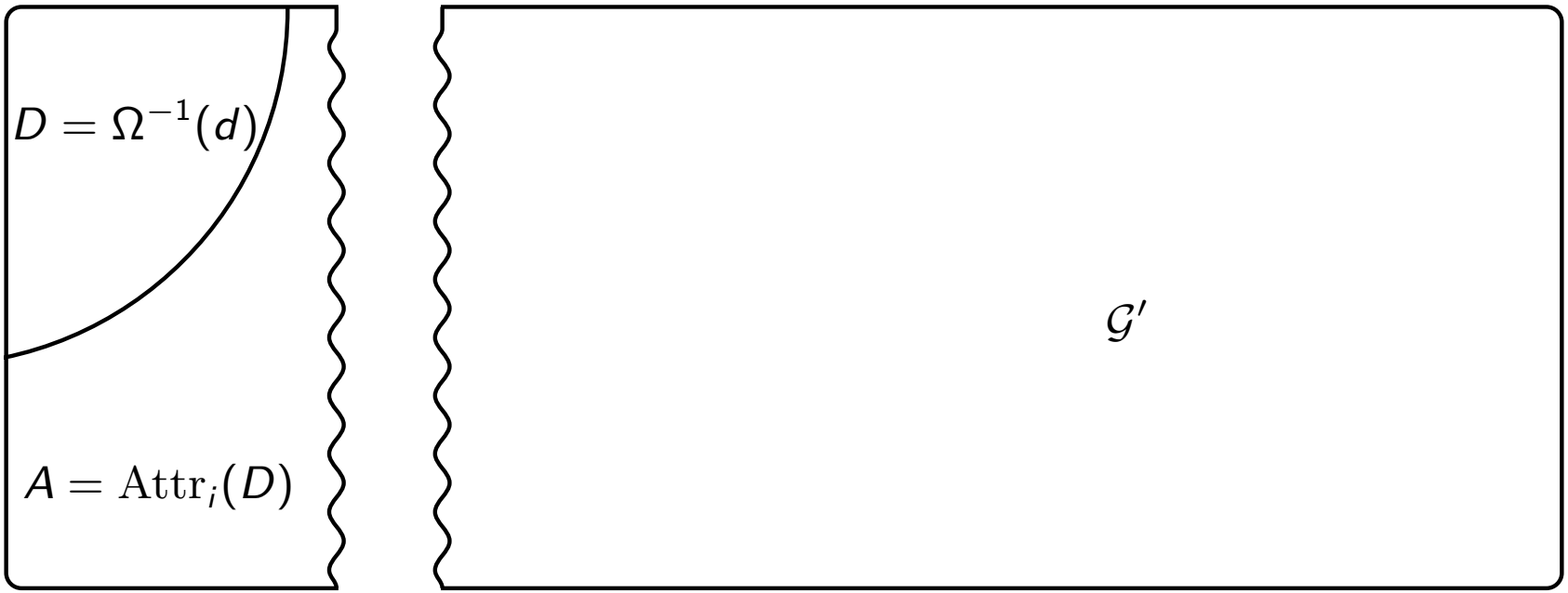


Now assume $A \neq V$.

Proof



Proof



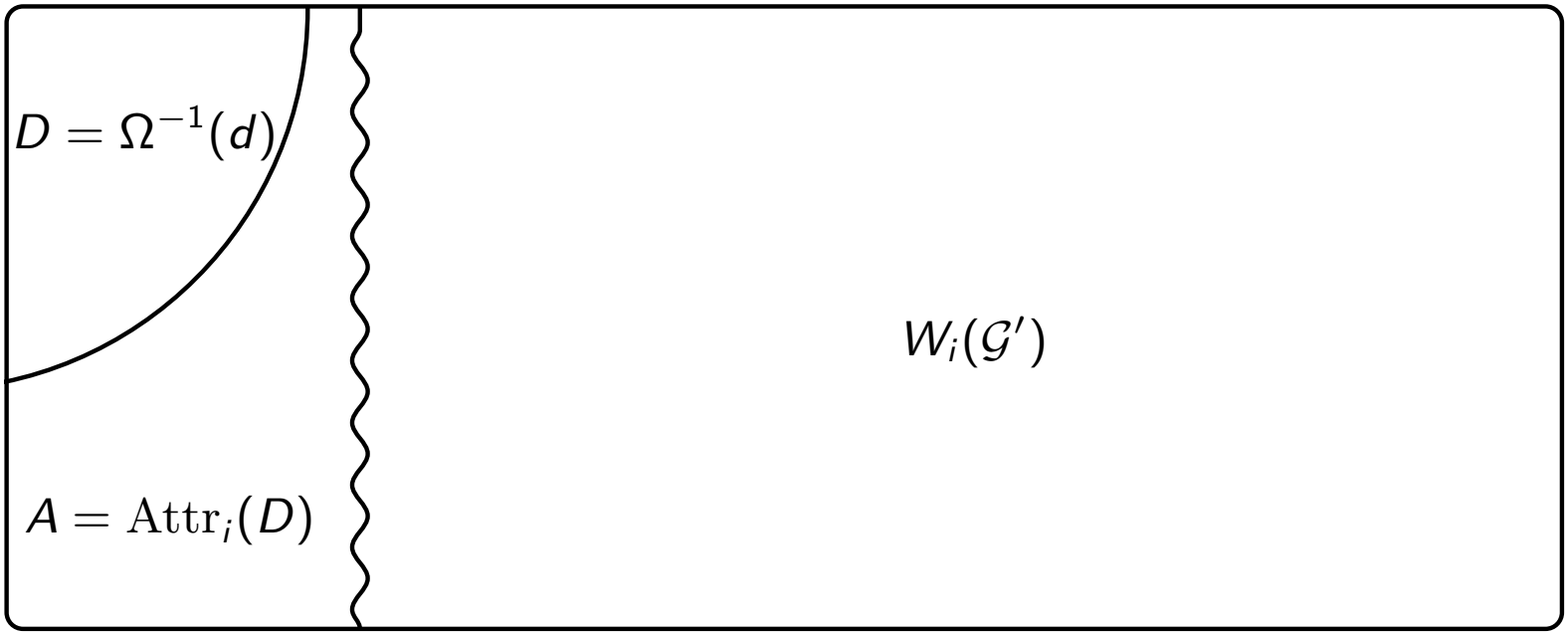
The induction hypothesis is applicable to $\mathcal{G}'$.

Proof

Proof

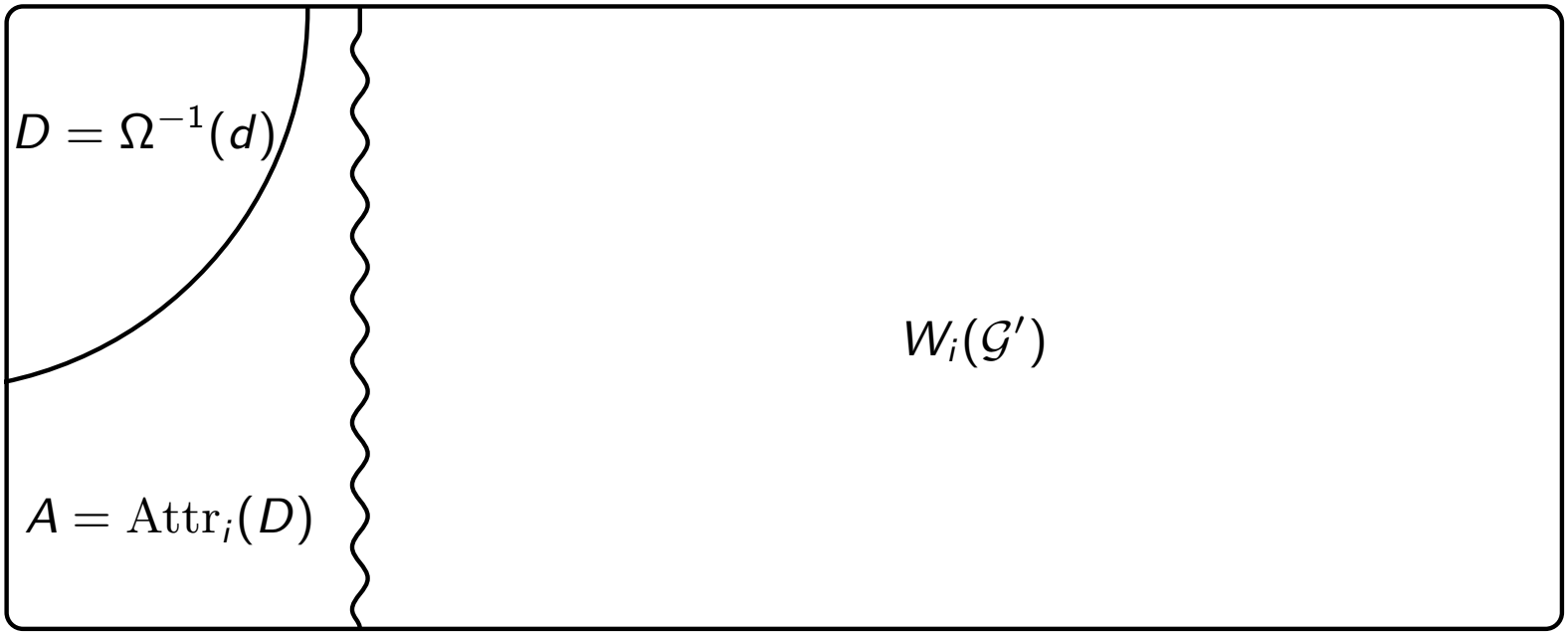
First, assume $W_{1-i}(\mathcal{G})$ is empty.

Proof



Claim $W_i(\mathcal{G}) = V$:

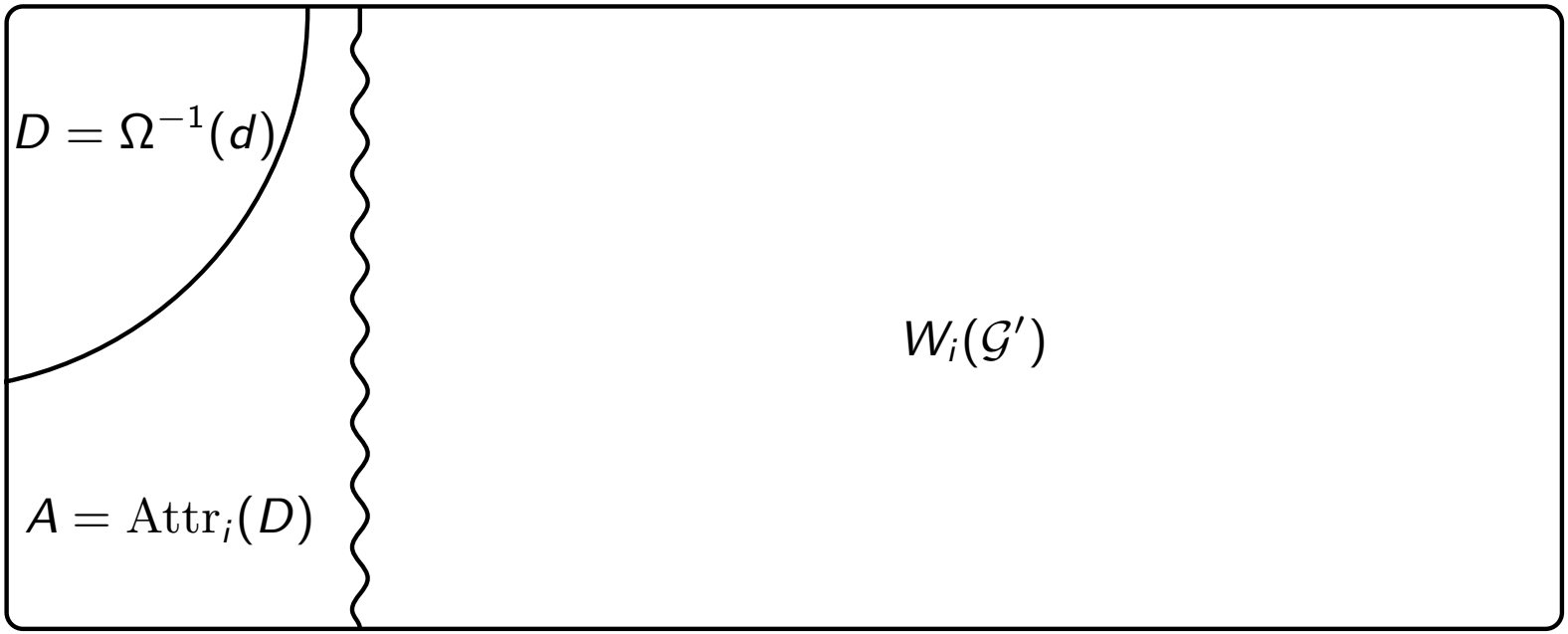
Proof



Claim $W_i(\mathcal{G}) = V$:

- In $W_i(\mathcal{G}')$ use winning strategy σ' from induction hypothesis.
- In $A \setminus D$ use attractor strategy.
- In D do anything.

Proof



Claim $W_i(\mathcal{G}) = V$:

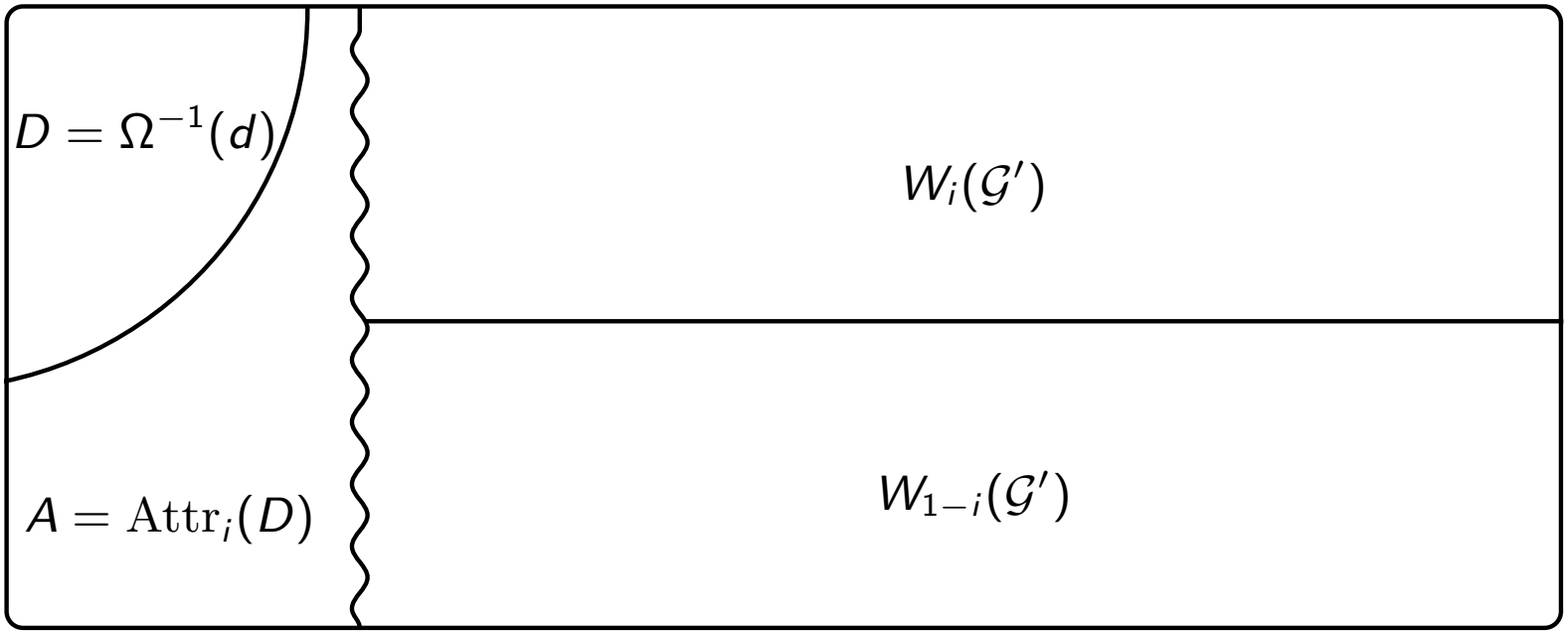
- In $W_i(\mathcal{G}')$ use winning strategy σ' from induction hypothesis.
- In $A \setminus D$ use attractor strategy.
- In D do anything.

Every consistent play either

- visits D infinitely often, or
- has a suffix that is consistent with σ'.

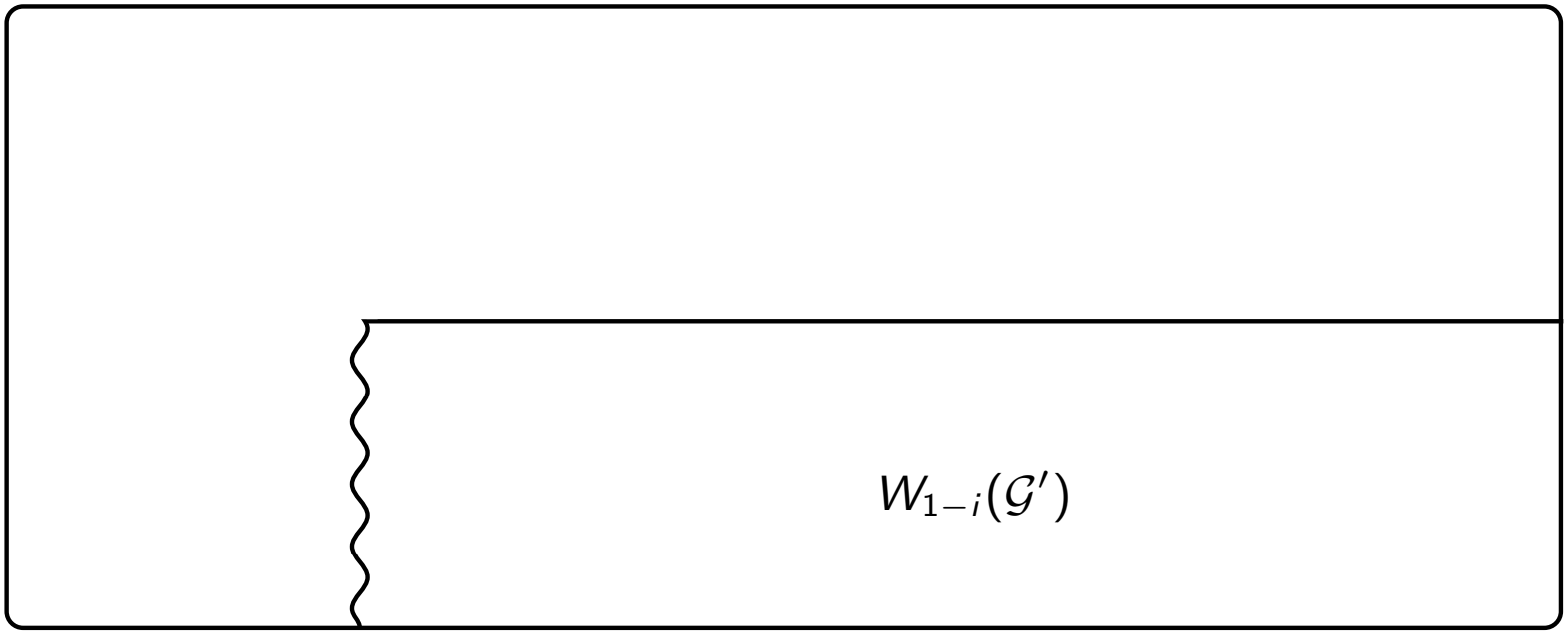
In both cases, Player i wins.

Proof

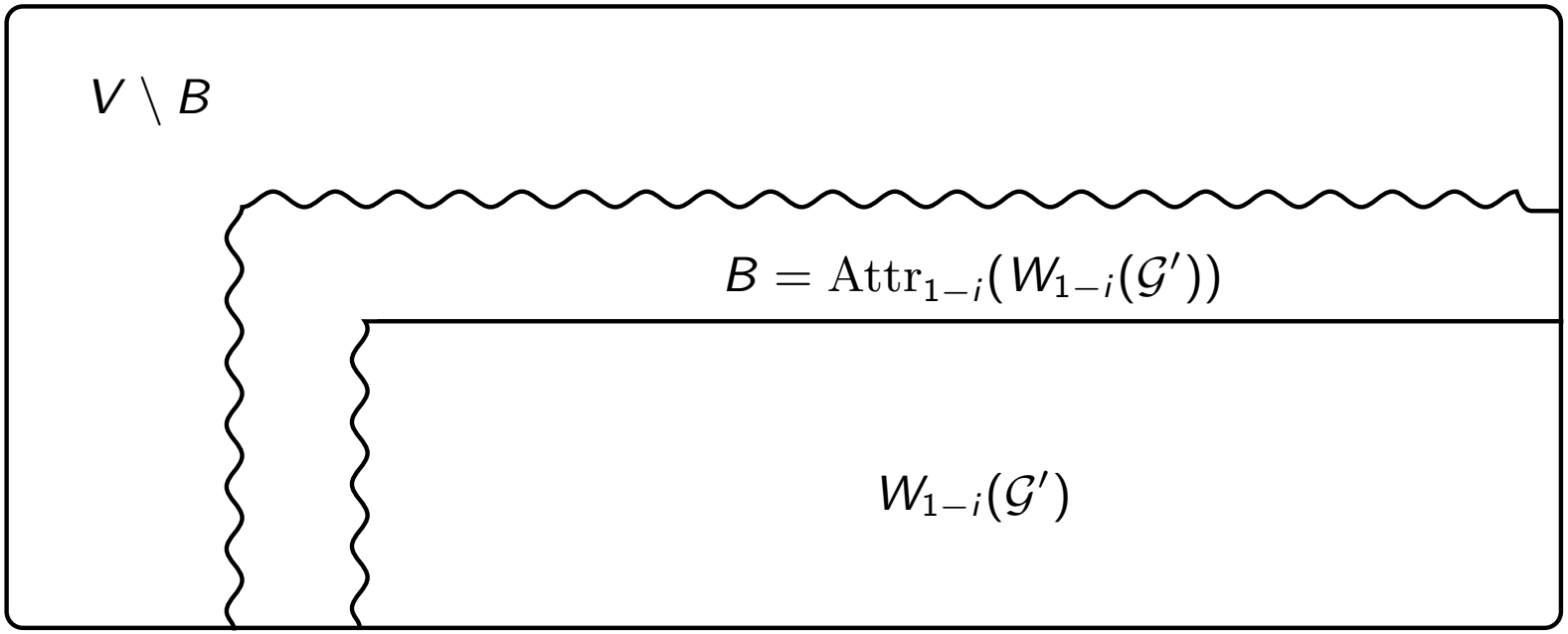


Now, assume $W_{1-i}(\mathcal{G})$ is non-empty.

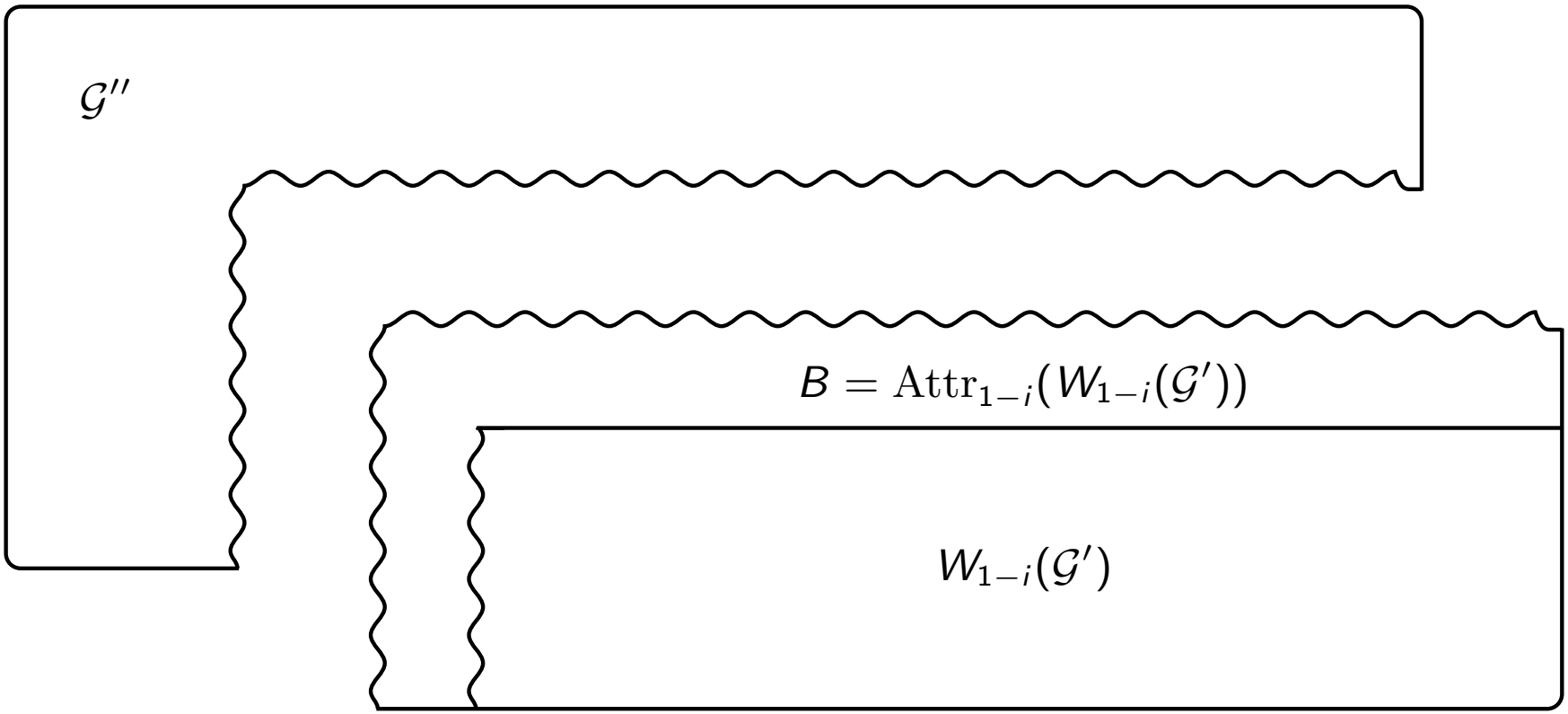
Proof



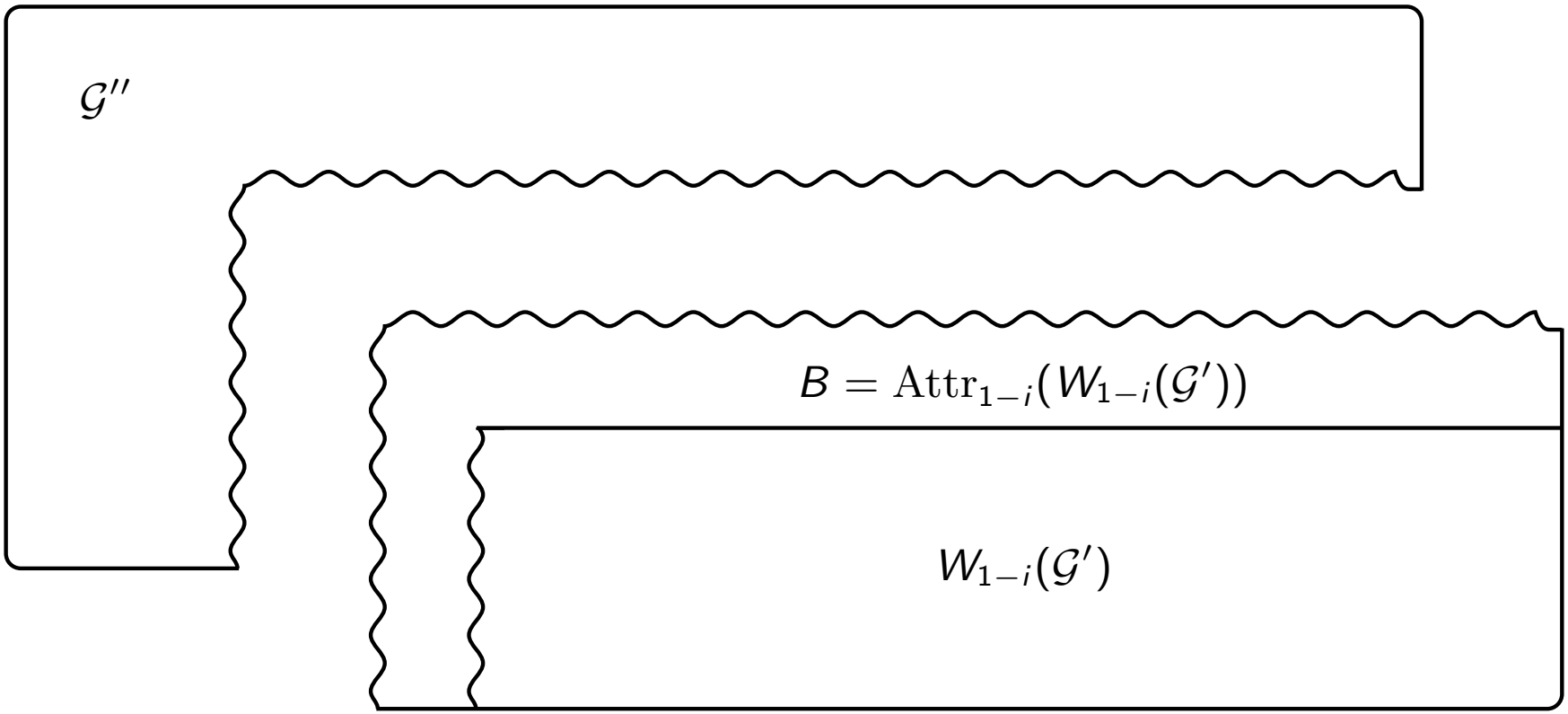
Proof



Proof

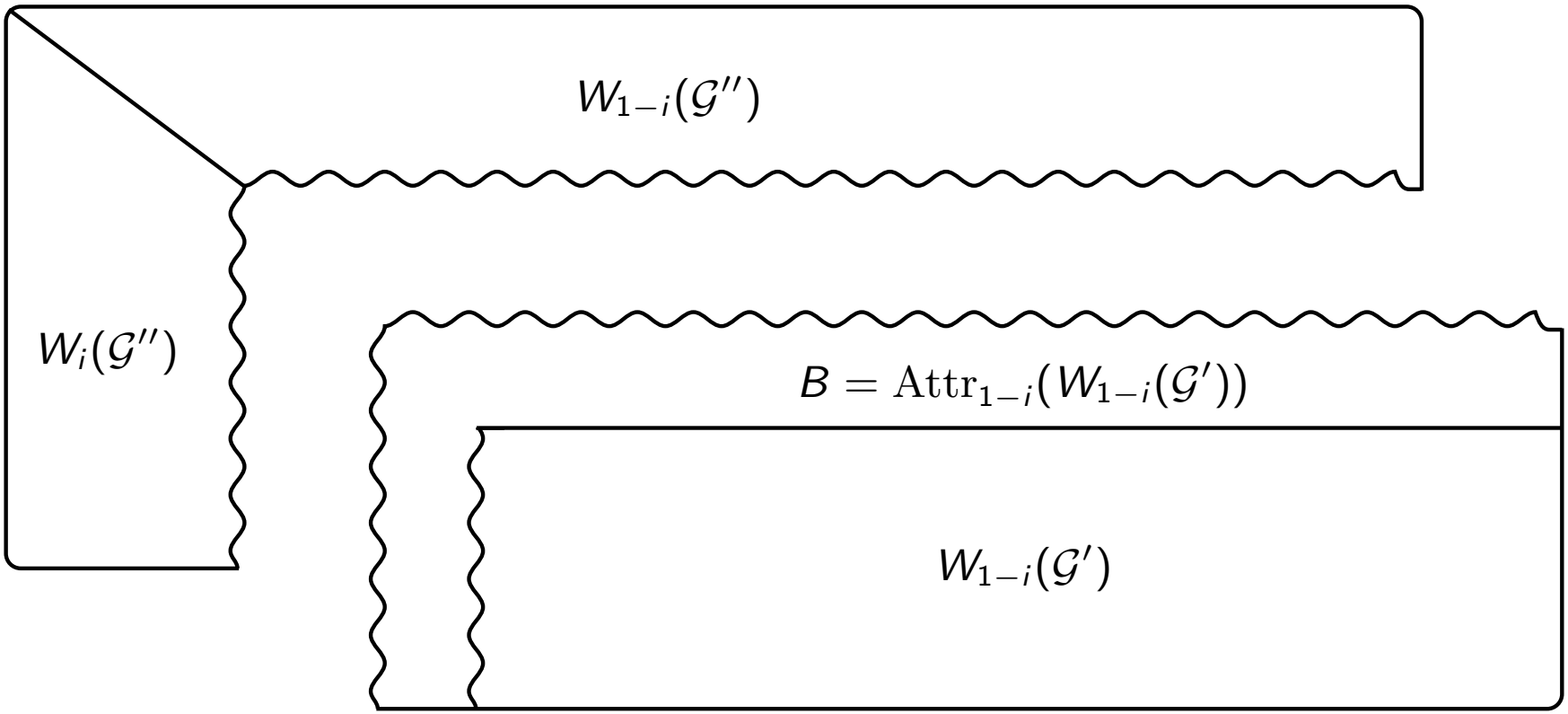


Proof

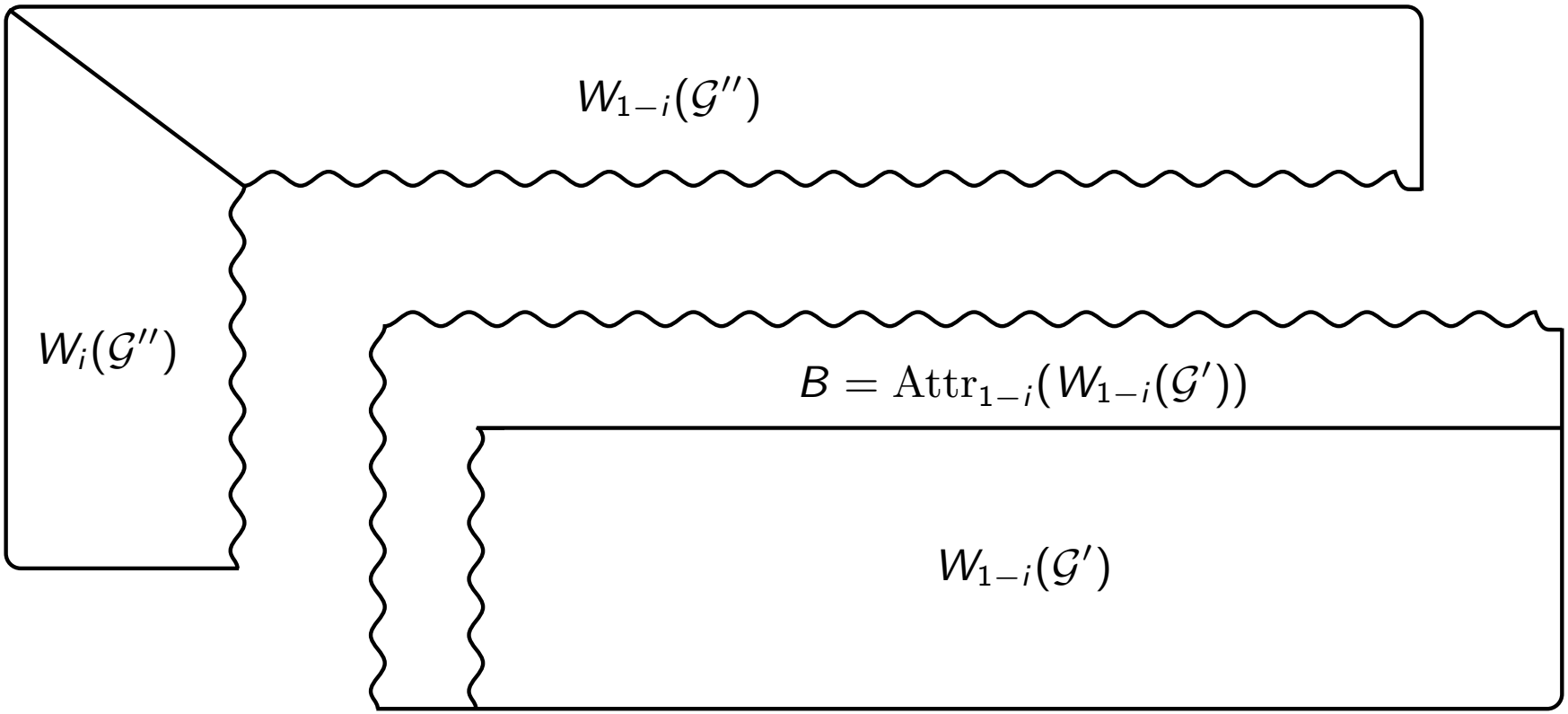


The induction hypothesis is applicable to $\mathcal{G}''$.

Proof

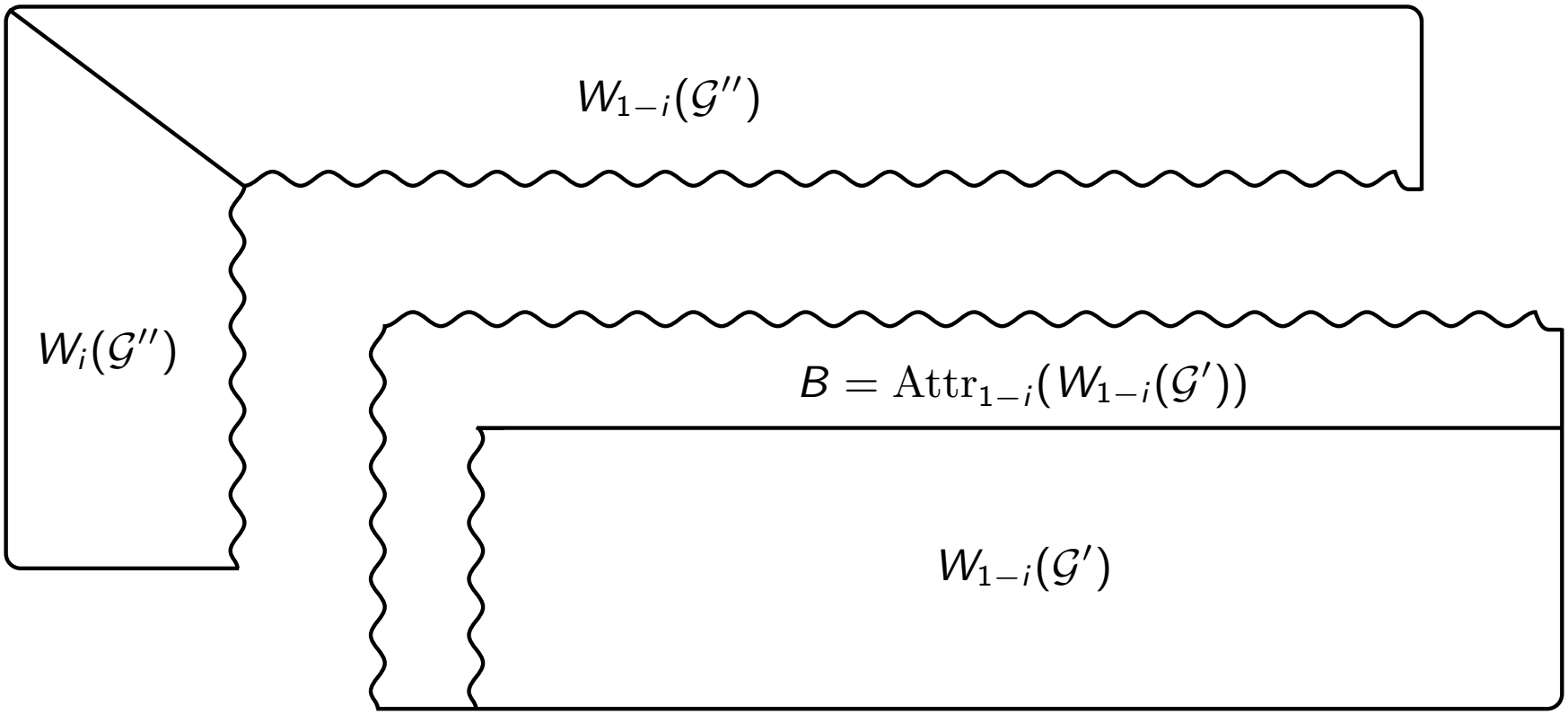

$W_{1-i}(\mathcal{G}'')$
$W_i(\mathcal{G}'')$
$B = \mathrm{Attr}_{1-i}(W_{1-i}(\mathcal{G}'))$
$W_{1-i}(\mathcal{G}')$

Proof



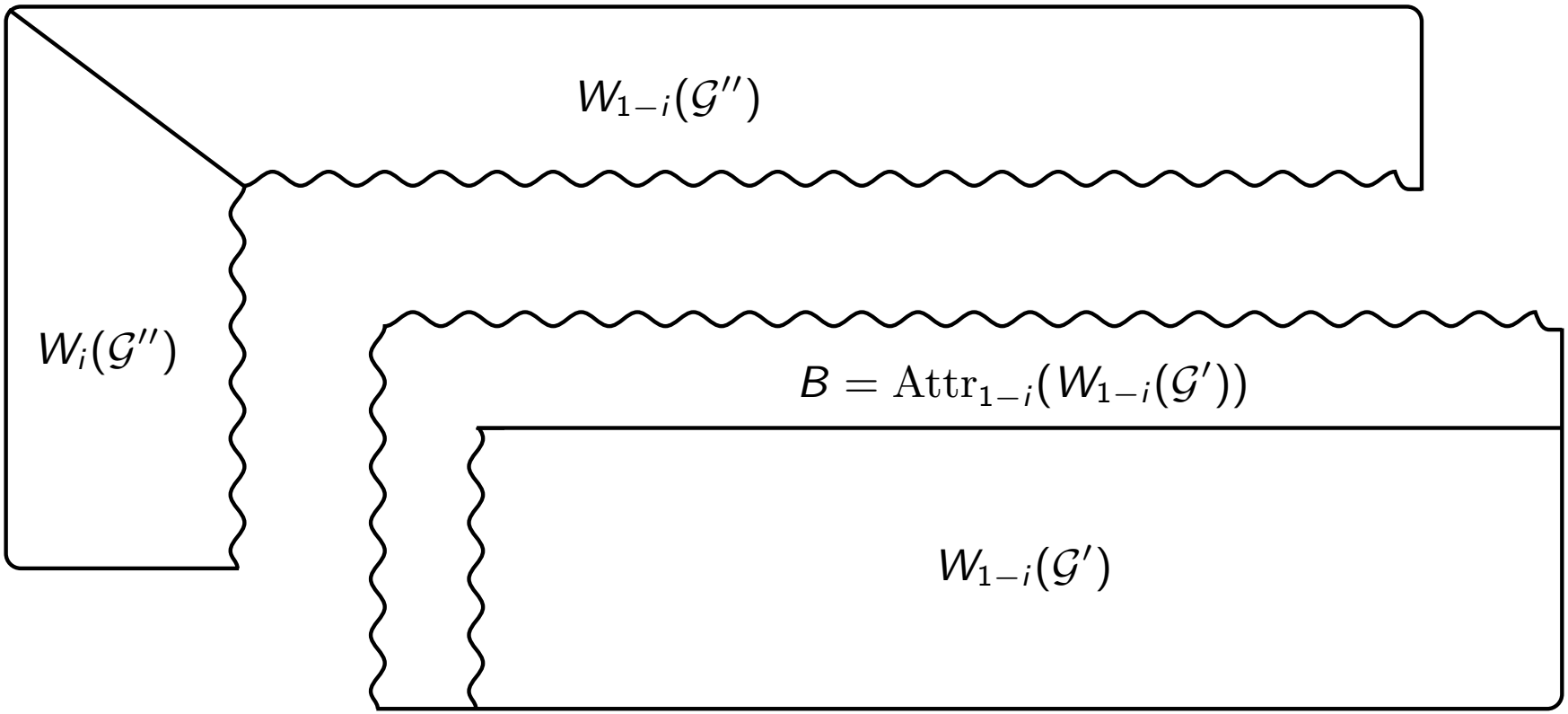
Claim $W_i(\mathcal{G}) = W_i(\mathcal{G}'')$:

Proof



Claim $W_i(\mathcal{G}) = W_i(\mathcal{G}'')$:
In $W_i(\mathcal{G}'')$ use winning strategy from induction hypothesis.

Proof

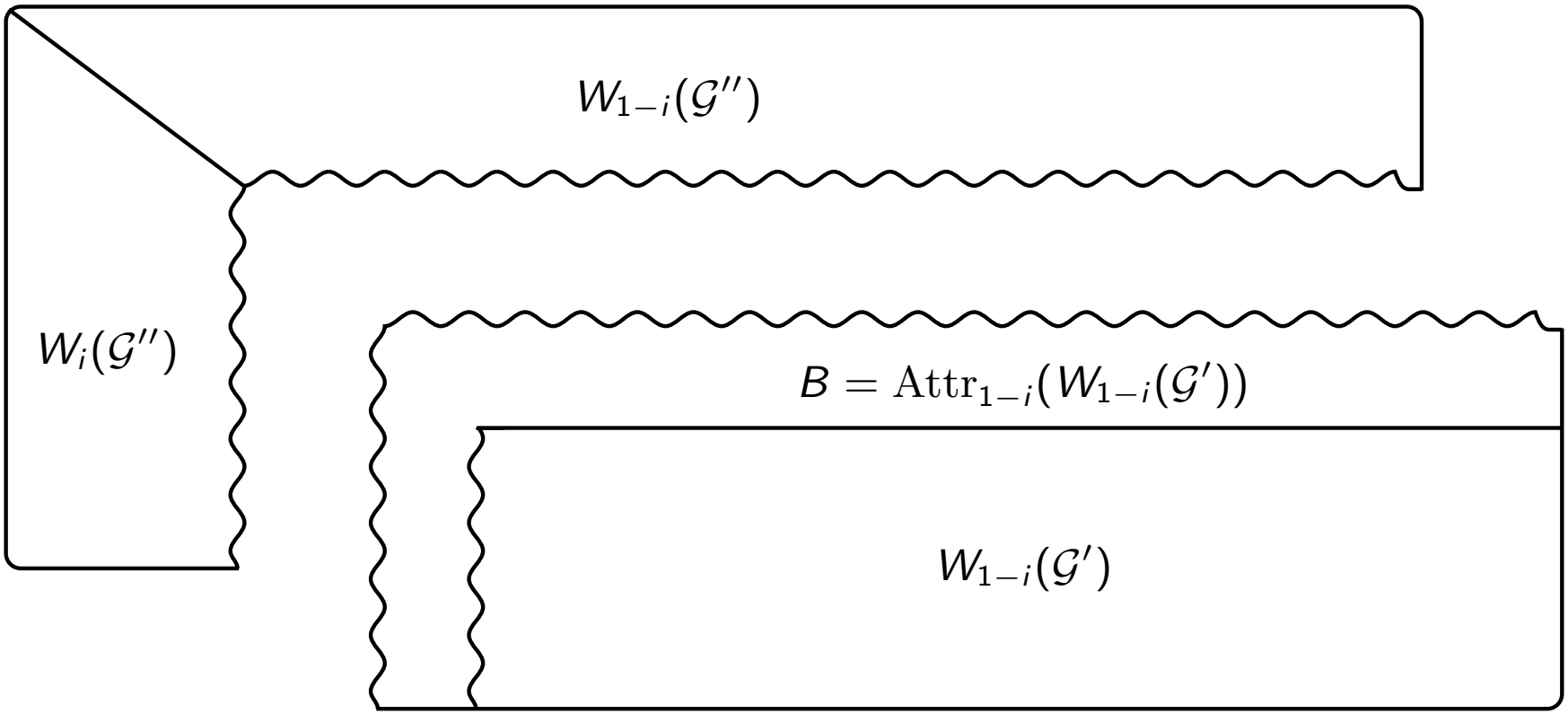


Claim $W_i(\mathcal{G}) = W_i(\mathcal{G}'')$:

In $W_i(\mathcal{G}'')$ use winning strategy from induction hypothesis.

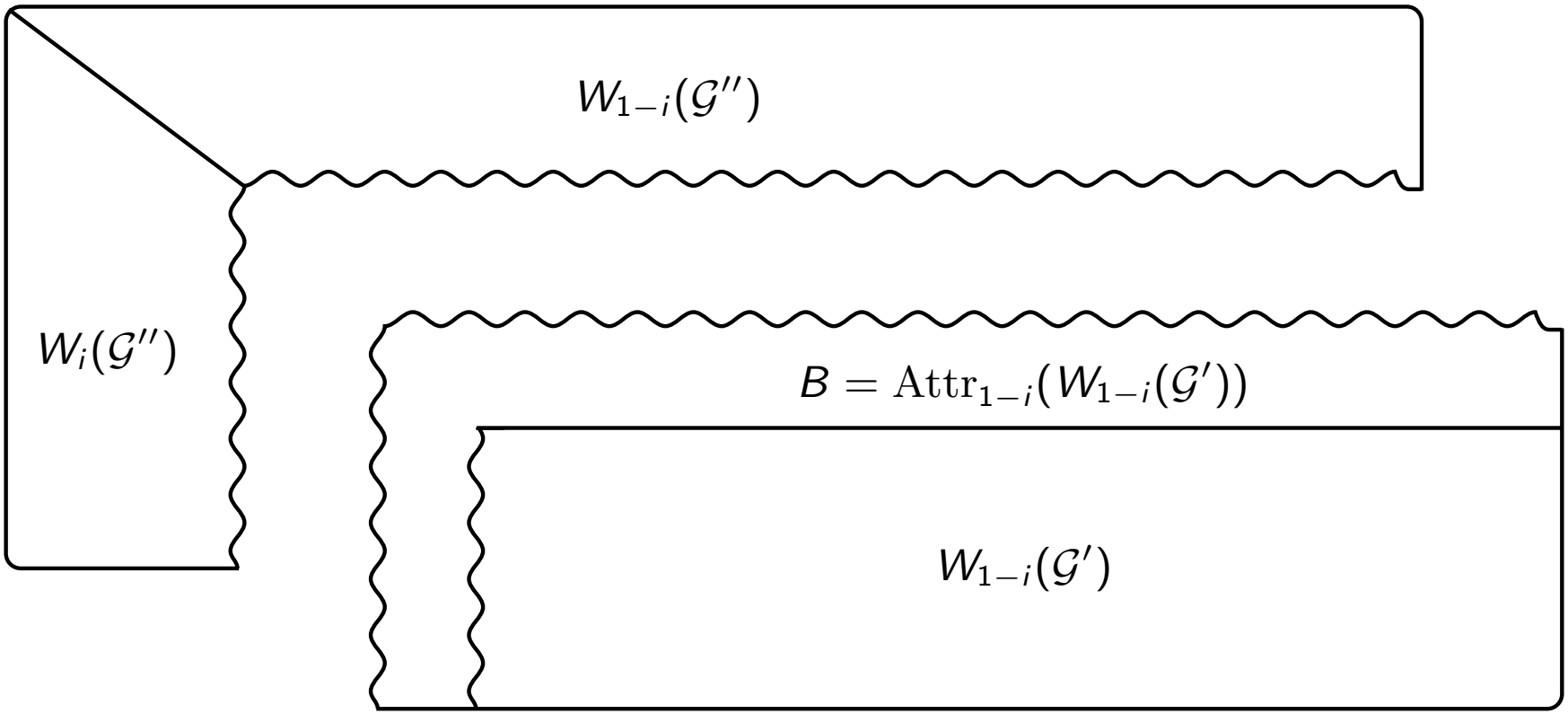
- $W_i(\mathcal{G}'')$ is a trap for Player $i-1$ in $\mathcal{A}$.
- Hence, σ' is also winning from $W_i(\mathcal{G}'')$ in $\mathcal{G}$.

Proof



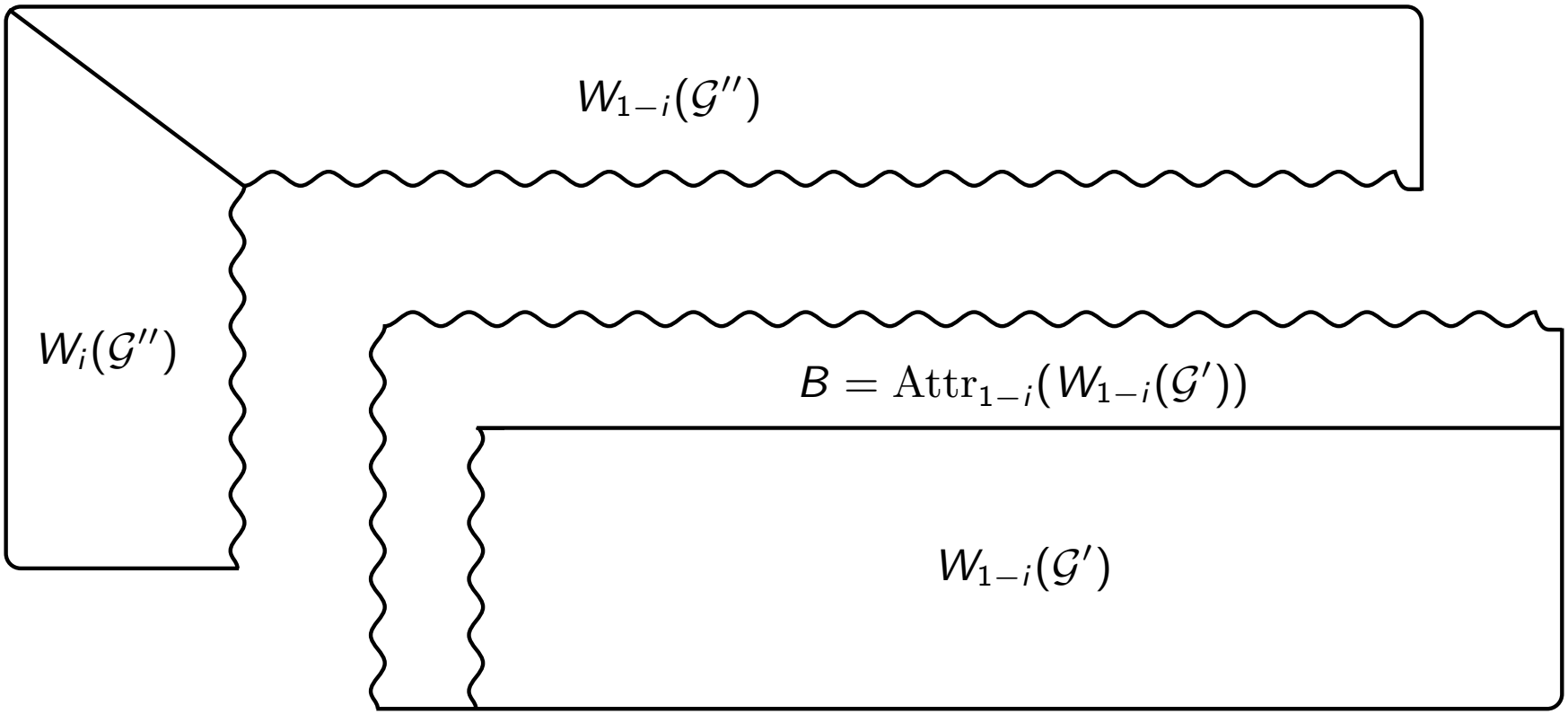
Claim $W_{i-1}(\mathcal{G}) = W_{1-i}(\mathcal{G}'') \cup B$:

Proof



Claim $W_{i-1}(\mathcal{G}) = W_{1-i}(\mathcal{G}'') \cup B$:
In $W_i(\mathcal{G}')$ and $W_i(\mathcal{G}'')$ use winning strategies σ' and σ'' from induction hypothesis, on attractor use attractor strategy.

Proof



Claim $W_{i-1}(\mathcal{G}) = W_{1-i}(\mathcal{G}'') \cup B$:
In $W_i(\mathcal{G}')$ and $W_i(\mathcal{G}'')$ use winning strategies σ' and σ'' from induction hypothesis, on attractor use attractor strategy.

- Each such play starting in $W_{1-i}(\mathcal{G}'') \cup B$ has a suffix that is consistent with σ' or σ''. Hence, it is winning for Player $1 - i$.

Algorithms for Parity Games

- Determinacy proof yields recursive algorithm with exponential running time.

Algorithms for Parity Games

- Determinacy proof yields recursive algorithm with exponential running time.
- Intriguing complexity-theoretic status: in $\mathrm{NP} \cap \mathrm{Co\text{-}NP}$ and thus unlikely to be complete for NP or $\mathrm{Co\text{-}NP}$).
- Even in $\mathrm{UP} \cap \mathrm{Co\text{-}UP}$ and other smaller complexity classes ("the easiest problem not in P").

Algorithms for Parity Games

- Determinacy proof yields recursive algorithm with exponential running time.
- Intriguing complexity-theoretic status: in $\mathrm{NP} \cap \mathrm{Co\text{-}NP}$ and thus unlikely to be complete for NP or $\mathrm{Co\text{-}NP}$).
- Even in $\mathrm{UP} \cap \mathrm{Co\text{-}UP}$ and other smaller complexity classes ("the easiest problem not in P").
- Until recently, the best deterministic algorithms had running times $\mathcal{O}(m \cdot n^{\frac{d}{3}})$ or $n^{\mathcal{O}(\sqrt{n})}$.
- But: in practice, the recursive algorithm is typically the fastest.

Algorithms for Parity Games

- Determinacy proof yields recursive algorithm with exponential running time.
- Intriguing complexity-theoretic status: in $\mathrm{NP} \cap \mathrm{Co\text{-}NP}$ and thus unlikely to be complete for NP or $\mathrm{Co\text{-}NP}$).
- Even in $\mathrm{UP} \cap \mathrm{Co\text{-}UP}$ and other smaller complexity classes ("the easiest problem not in P").
- Until recently, the best deterministic algorithms had running times $\mathcal{O}(m \cdot n^{\frac{d}{3}})$ or $n^{\mathcal{O}(\sqrt{n})}$.
- But: in practice, the recursive algorithm is typically the fastest.
- **Open problem:** is solving parity games in polynomial time?

A Breakthrough

Theorem

Parity games can be solved in quasi-polynomial time, i.e., in time $\mathcal{O}(n^{\log d+6})$.

The following is based on slides by John Fearnley (University of Liverpool).

Intuition

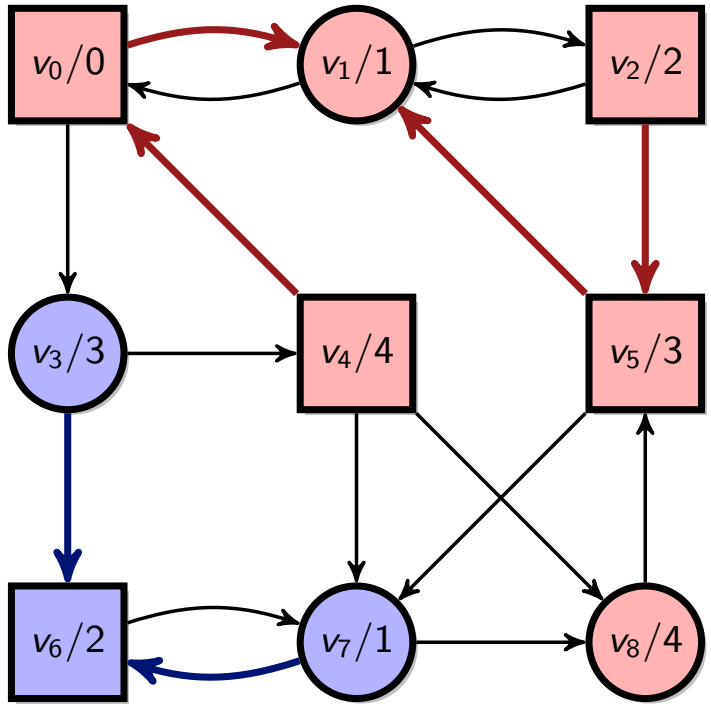
Consider the following "finite-duration" variant of parity games:

- The players move a token through the arena until a cycle is closed.
- Player 0 wins if and only if the maximal color on the cycle is even.

Intuition

Consider the following "finite-duration" variant of parity games:

- The players move a token through the arena until a cycle is closed.
- Player 0 wins if and only if the maximal color on the cycle is even.



Intuition

Consider the following "finite-duration" variant of parity games:

- The players move a token through the arena until a cycle is closed.
- Player 0 wins if and only if the maximal color on the cycle is even.

Lemma

Player 0 *wins a parity game from* v *if and only if she wins the finite-duration variant from* v*.*

Proof.

By positional determinacy. □

Intuition

The finite-duration game is essentially a reachability game:

- Vertices are cycle-free play prefixes.
- Once a cycle is closed, the play ends in an accepting sink if the maximal color on the cycle is even. Otherwise, it ends in a rejecting sink.
- Player 0 wins if the accepting sink is reached.

Intuition

The finite-duration game is essentially a reachability game:

- Vertices are cycle-free play prefixes.
- Once a cycle is closed, the play ends in an accepting sink if the maximal color on the cycle is even. Otherwise, it ends in a rejecting sink.
- Player 0 wins if the accepting sink is reached.

Problem

The reachability game has n^n vertices in the worst case.

Intuition

The finite-duration game is essentially a reachability game:

- Vertices are cycle-free play prefixes.
- Once a cycle is closed, the play ends in an accepting sink if the maximal color on the cycle is even. Otherwise, it ends in a rejecting sink.
- Player 0 wins if the accepting sink is reached.

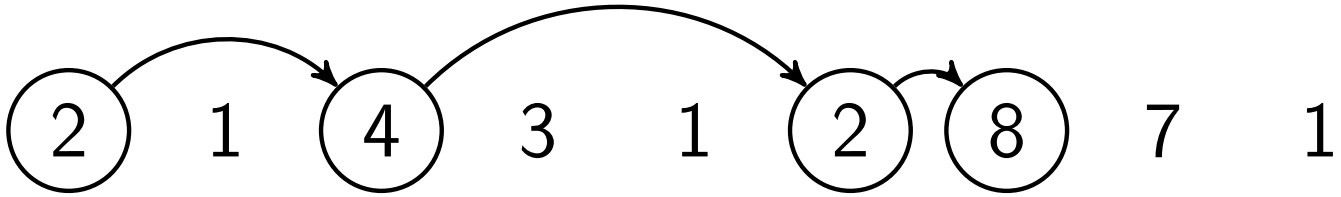
Problem

The reachability game has n^n vertices in the worst case.

The quasi-polynomial time algorithm is based on the following idea:

- Save resources by not aiming for the first cycle.
- Instead find a witness that is "easier" to check while still being complete.

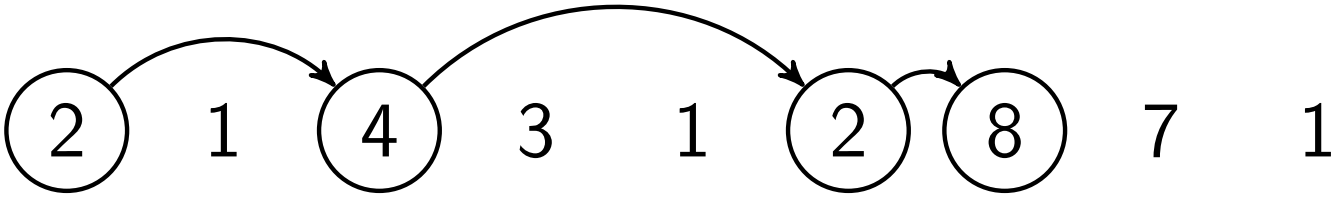
i-sequences



An i-sequence is a set of 2^i positions such that

- **Evenness:** every position has an even color except (possibly) the last,
- **Inner domination:** every subsequence is dominated by one of its end points, and
- **Outer domination:** the final position dominates the rest of the sequence

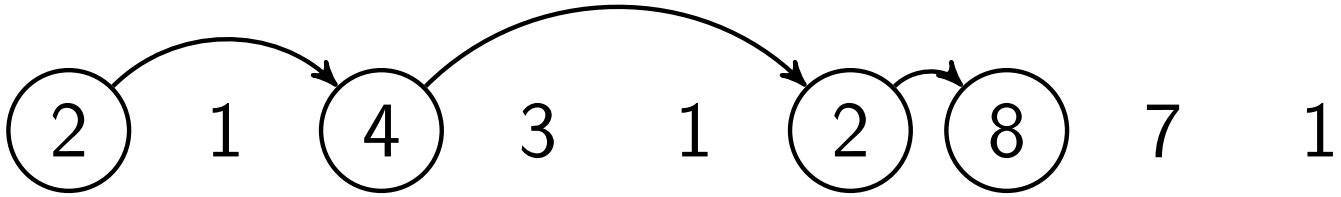
i-sequences



Intuition

- Player 0 wins $\Rightarrow$ we should see a $\log(n+1)$-sequence.
  - Take each position to be the largest color seen infinitely often.

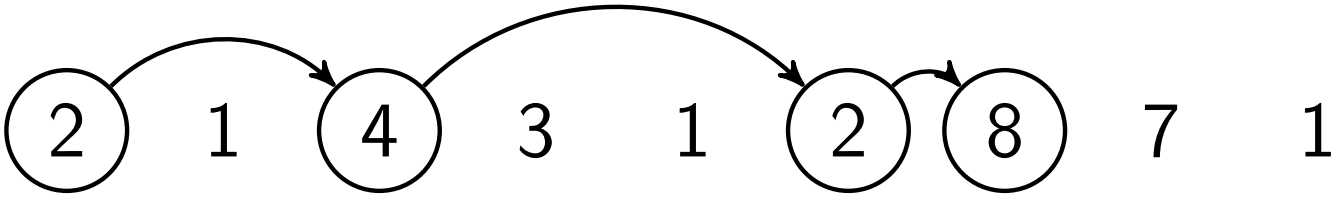
i-sequences



Intuition

- Player 0 wins $\Rightarrow$ we should see a $\log(n+1)$-sequence.
  - Take each position to be the largest color seen infinitely often.
- Player 1 wins $\Rightarrow$ we should not see a $\log(n+1)$-sequence.
  - The largest color seen infinitely often is odd.
  - This color cannot be inner dominated.
  - Player 1 can force it to be seen before move n.

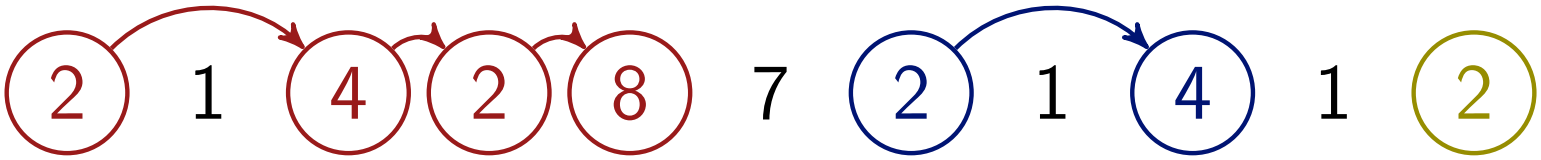
i-sequences



The idea behind the algorithm:

- Watch the players play the game.
- Uses a compact data structure to track the longest i-sequence.
- If we see a $(\log n)$-sequence, declare Player 0 to be the winner.

The Data Structure

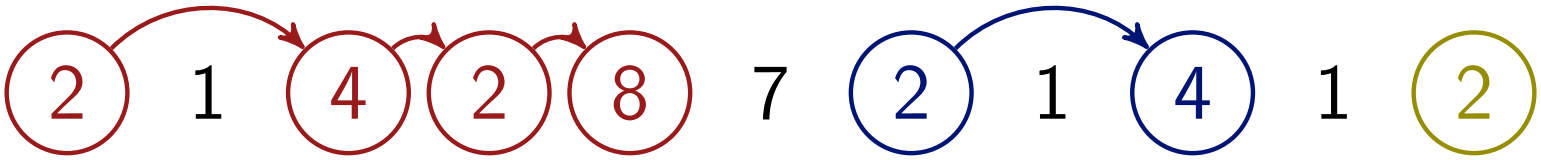


– 8 4 2

Each component of the data structure is either – or a color.

- **Sequences:** If component i is not – then
  - the play contains an i-sequence and
  - the component records the outer domination color
- **Order:** The sequences are ordered, i.e,
  - the 1-sequence comes after the 2-sequence, etc.

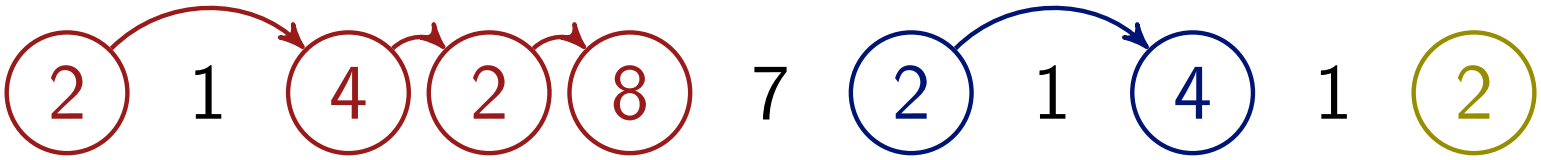
The Data Structure



$-\ 8\ 4\ 2$

- The data structure has $\log n$ components.
- Each component needs $\log(d+1)$ bits.
- So the total size is $(\log n)(\log(d+1)) \in \mathcal{O}(\log^2 n)$.

The Data Structure



– 8 4 2

We will feed a play into the data structure.

- We start with – – – –.
- Each time we see a new color, we use an update rule...

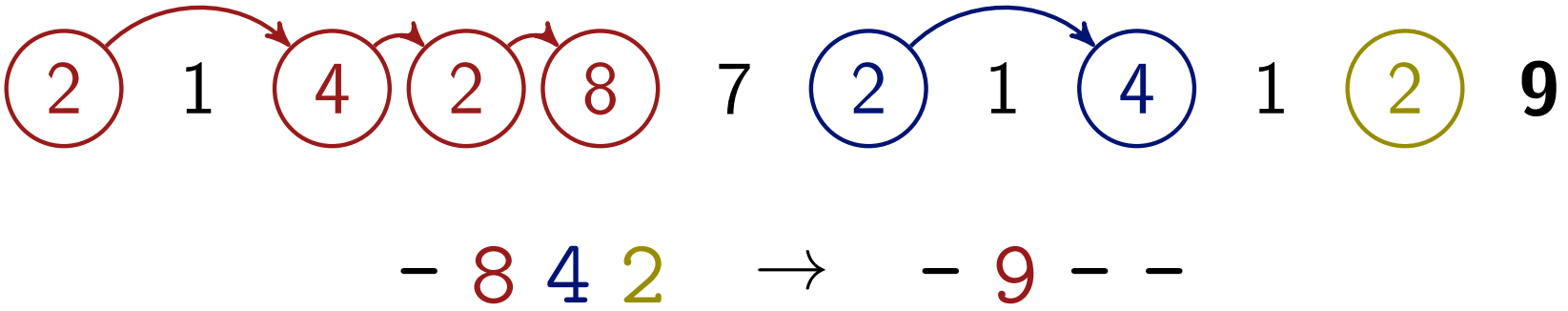
The Data Structure

$$-\ 8\ 4\ 2 \quad \rightarrow \quad 4\ -\ -\ -$$

Type 1 Update: New color c

- Find the largest i such that
  - position i is blank or $c <$ color at position i and
  - all colors from position i to the right are even.
- Then:
  - Put the new color at position i.
  - Blank all lower positions.

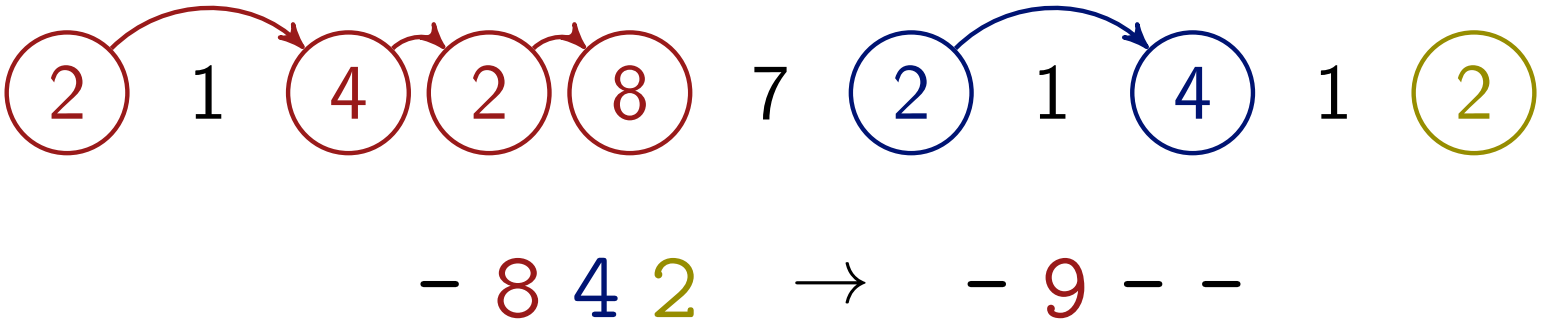
The Data Structure



Type 2 Update: New color c

- Find the largest i such that $c >$ color at position i.
- Then:
  - Put the new color at position i.
  - Blank all lower positions.

The Data Structure



Type 2 Update: New color c

- Find the largest i such that $c >$ color at position i.
- Then:
  - Put the new color at position i.
  - Blank all lower positions.

First apply Type 1, then apply Type 2

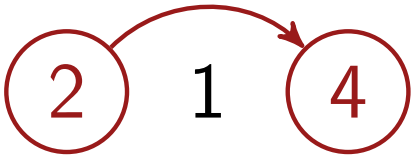
An Example Play

- - - 2

An Example Play

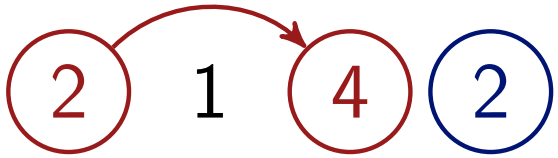

```
    - - - 2
→   - - 1 -
```

An Example Play




```
    - - - 2
→   - - 1 -
→   - - 4 -
```

An Example Play




```
    - - - 2
→   - - 1 -
→   - - 4 -
→   - - 4 2
```

An Example Play


```
    - - - 2
→   - - 1 -
→   - - 4 -
→   - - 4 2
→   - 8 - -
```

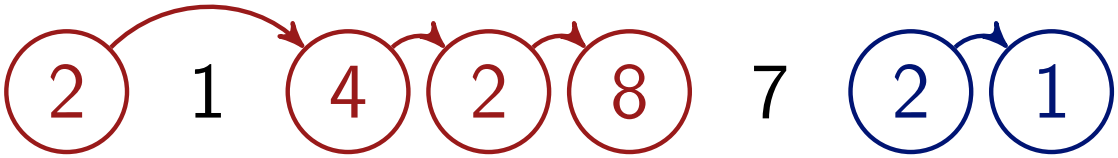
An Example Play


```
    - - - 2
→   - - 1 -
→   - - 4 -
→   - - 4 2
→   - 8 - -
→   - 8 - 7
```

An Example Play

$$
\begin{array}{ll}
 & - \; - \; - \; 2 \\
\rightarrow & - \; - \; 1 \; - \\
\rightarrow & - \; - \; 4 \; - \\
\rightarrow & - \; - \; 4 \; 2 \\
\rightarrow & - \; 8 \; - \; - \\
\rightarrow & - \; 8 \; - \; 7 \\
\rightarrow & - \; 8 \; - \; 2
\end{array}
$$

An Example Play



$$
\begin{array}{ll}
 & - \; - \; - \; 2 \\
\rightarrow & - \; - \; 1 \; - \\
\rightarrow & - \; - \; 4 \; - \\
\rightarrow & - \; - \; 4 \; 2 \\
\rightarrow & - \; 8 \; - \; - \\
\rightarrow & - \; 8 \; - \; 7 \\
\rightarrow & - \; 8 \; - \; 2 \\
\rightarrow & - \; 8 \; 1 \; -
\end{array}
$$

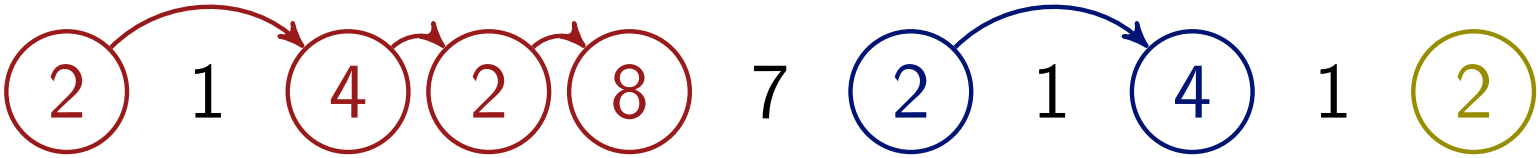
An Example Play


```
    - - - 2
→   - - 1 -
→   - - 4 -
→   - - 4 2
→   - 8 - -
→   - 8 - 7
→   - 8 - 2
→   - 8 1 -
→   - 8 4 -
```

An Example Play

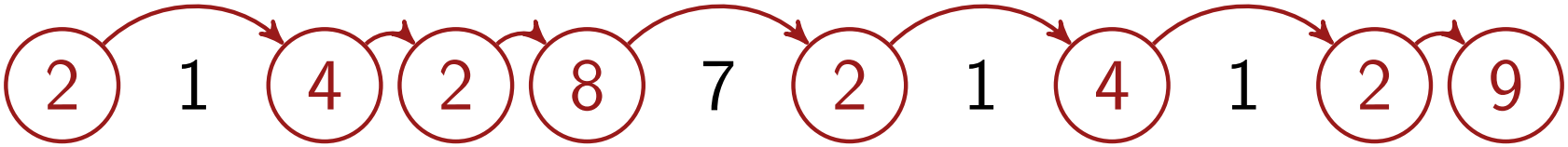

```
    - - - 2
→   - - 1 -
→   - - 4 -
→   - - 4 2
→   - 8 - -
→   - 8 - 7
→   - 8 - 2
→   - 8 1 -
→   - 8 4 -
→   - 8 4 1
```

An Example Play



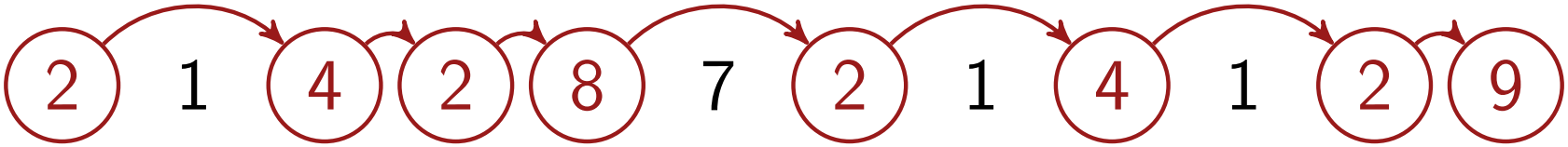

```
    - - - 2
→   - - 1 -
→   - - 4 -
→   - - 4 2
→   - 8 - -
→   - 8 - 7
→   - 8 - 2
→   - 8 1 -
→   - 8 4 -
→   - 8 4 1
→   - 8 4 2
```

An Example Play




```
    - - - 2
→   - - 1 -
→   - - 4 -
→   - - 4 2
→   - 8 - -
→   - 8 - 7
→   - 8 - 2
→   - 8 1 -
→   - 8 4 -
→   - 8 4 1
→   - 8 4 2
→   9 - - -
```

An Example Play




```
    - - - 2
→   - - 1 -
→   - - 4 -
→   - - 4 2
→   - 8 - -
→   - 8 - 7
→   - 8 - 2
→   - 8 1 -
→   - 8 4 -
→   - 8 4 1
→   - 8 4 2
→   9 - - -
→   ...
```

The Algorithm

Use the data structure as memory structure and play a reachability game in the product arena.

- Initialize memory with – – – – and use update rules to implement update function.
- Goal vertices: those whose first position is set.
- The resulting game has $n^{\mathcal{O}(\log n)}$ vertices and can be constructed and solved in time $n^{\mathcal{O}(\log n)}$.

The Algorithm

Use the data structure as memory structure and play a reachability game in the product arena.

- Initialize memory with – – – – and use update rules to implement update function.
- Goal vertices: those whose first position is set.
- The resulting game has $n^{\mathcal{O}(\log n)}$ vertices and can be constructed and solved in time $n^{\mathcal{O}(\log n)}$.

Lemma

Player 0 *wins the parity game from* v *if and only if she wins the reachability game from* $(v, - \ - \ - \ -)$.

The Algorithm

Use the data structure as memory structure and play a reachability game in the product arena.

- Initialize memory with $-\ -\ -\ -$ and use update rules to implement update function.
- Goal vertices: those whose first position is set.
- The resulting game has $n^{\mathcal{O}(\log n)}$ vertices and can be constructed and solved in time $n^{\mathcal{O}(\log n)}$.

Lemma

Player 0 *wins the parity game from* v *if and only if she wins the reachability game from* $(v, -\ -\ -\ -)$.

Note

This is not a game reduction as introduced in the previous lecture!

The Latest

[CJKLS 17] Quasi-polynomial time and space.

The Latest

[CJKLS 17] Quasi-polynomial time and space.

[GI-J 17] A simpler proof, slightly better running time, and finer running time analysis.

The Latest

[CJKLS 17] Quasi-polynomial time and space.

[GI-J 17] A simpler proof, slightly better running time, and finer running time analysis.

[JL 17] Quasi-polynomial time and linear space with a completely different approach and a fine running time analysis.

The Latest

[CJKLS 17] Quasi-polynomial time and space.

[GI-J 17] A simpler proof, slightly better running time, and finer running time analysis.

[JL 17] Quasi-polynomial time and linear space with a completely different approach and a fine running time analysis.

[FJSW 17] Quasi-polynomial time and linear space by a refinement of the original approach.

The Latest

[CJKLS 17] Quasi-polynomial time and space.

[GI-J 17] A simpler proof, slightly better running time, and finer running time analysis.

[JL 17] Quasi-polynomial time and linear space with a completely different approach and a fine running time analysis.

[FJSW 17] Quasi-polynomial time and linear space by a refinement of the original approach.

But still no polynomial-time algorithm.