Games

Christof Löding

RWTH Aachen, Germany

MOVEP 2008, June 23–27, Orleans, France

Games

What kind of games do we consider?

- **two player**
- **zero-sum (win-lose)**
- **sequential or turn-based**
- **deterministic (no randomization/probabilities)**
- **perfect information**
- **infinite duration**

Origin

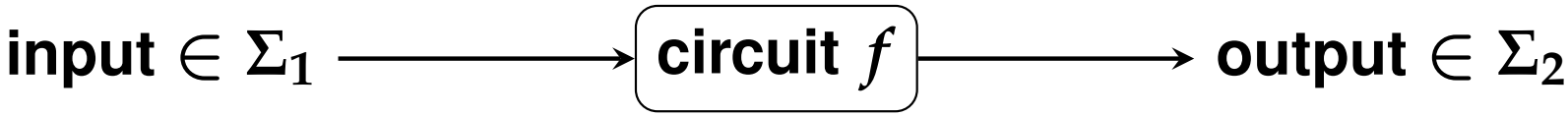
Circuit synthesis and Church's problem (1957)

Setting:

- **Sequence of input signals arrives**
- **Circuit produces a sequence of output signals (depending on the inputs it has seen)**
- **Result is a non-terminating sequence of input and output signals**
- **A logical specification describes the desired properties of these sequences**

Task: Automatically synthesize a circuit from the specification

More formally

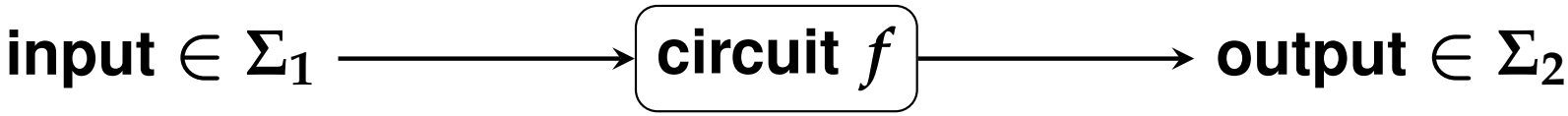


- Input sequence $\alpha \in \Sigma_1^\omega$ and output sequence $\beta \in \Sigma_2^\omega$
- Specification $\varphi(\alpha, \beta)$

Problem:

- Decide if the there is a sequential transformation $f : \Sigma_1^* \longrightarrow \Sigma_2$ realizing φ, and construct one if possible.

More formally



- Input sequence $\alpha \in \Sigma_1^\omega$ and output sequence $\beta \in \Sigma_2^\omega$
- Specification $\varphi(\alpha, \beta)$

Problem:

- Decide if the there is a sequential transformation $f : \Sigma_1^* \longrightarrow \Sigma_2$ realizing φ, and construct one if possible.

Modeled as a game:

- One player plays input signals, the other player output signals
- The specification is the winning condition for the output player
- A transformation f realizing the specification is a winning strategy for the output player

Reactive Systems

- Games can serve as general model for reactive systems, i.e., systems with input/output behavior
- To obtain more realistic models it is often required to add features like time, probabilities, concurrency, ...
- To study these more complex models a good understanding of the core theory originating from Church's problem is fundamental
  $\leadsto$ this tutorial

1. Basics
2. Positional Strategies: Reachability and Attractors
3. Büchi Games
4. Muller Games and Finite Memory Strategies
5. Parity Games
   - Positional Determinacy
   - Algorithms for Parity Games
   - Reducing Muller Games to Parity Games
6. Applications

1 Basics

2 Positional Strategies: Reachability and Attractors

3 Büchi Games

4 Muller Games and Finite Memory Strategies

5 Parity Games

- Positional Determinacy
- Algorithms for Parity Games
- Reducing Muller Games to Parity Games

6 Applications

Notations

For a set X:

- $|X| =$ **size of** X
- $X^* =$ **the set of finite sequences over** X
- $X^\omega =$ **the set of infinite sequences over** X
- **For** $\alpha \in X^\omega$**:**

$$\mathrm{Inf}(\alpha) = \{x \in X \mid x \text{ occurs infinitely often in } \alpha\}.$$

Game Graph / Arena

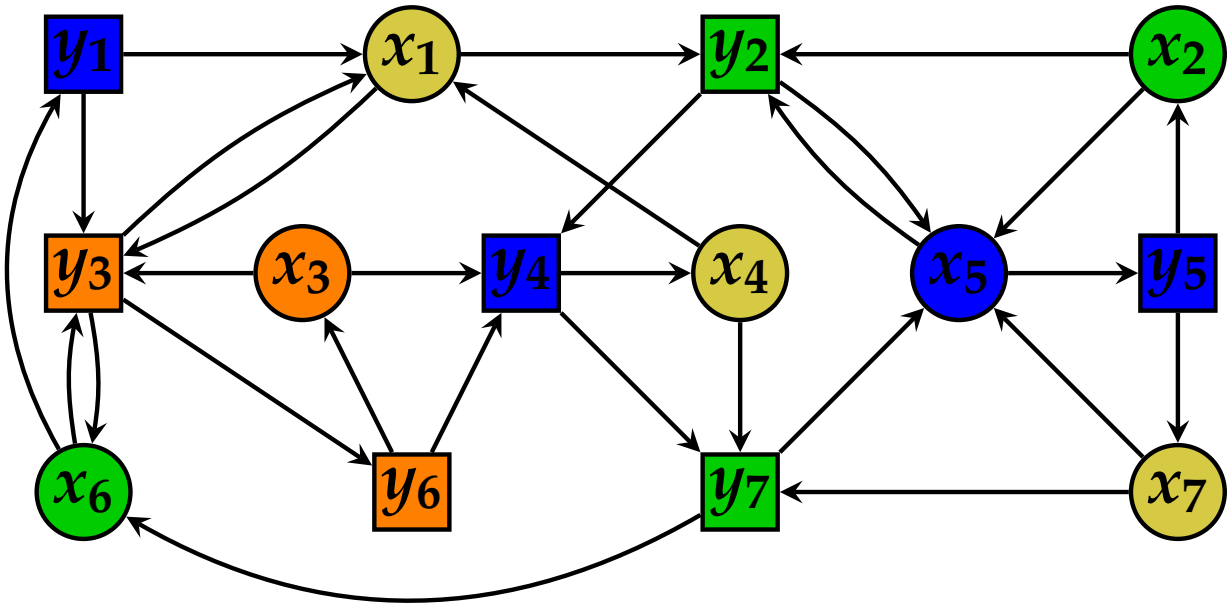
$G = (V_\exists, V_\forall, E, c)$

- $V_\exists$: **vertices of Eva (player 1, circle)**
- $V_\forall$: **vertices of Adam (player 2, box)**
- $E \subseteq V \times V$: **edges with** $V = V_\exists \cup V_\forall$
- $c : V \longrightarrow C$ **with a finite set of colors** C

Game Graph / Arena

$G = (V_\exists, V_\forall, E, c)$

- $V_\exists$: vertices of Eva (player 1, circle) $\{x_1, \ldots, x_7\}$
- $V_\forall$: vertices of Adam (player 2, box) $\{y_1, \ldots, y_7\}$
- $E \subseteq V \times V$: edges with $V = V_\exists \cup V_\forall$ $\ldots$
- $c : V \longrightarrow C$ with a finite set of colors C $\{\bullet, \bullet, \bullet, \bullet\}$



Plays

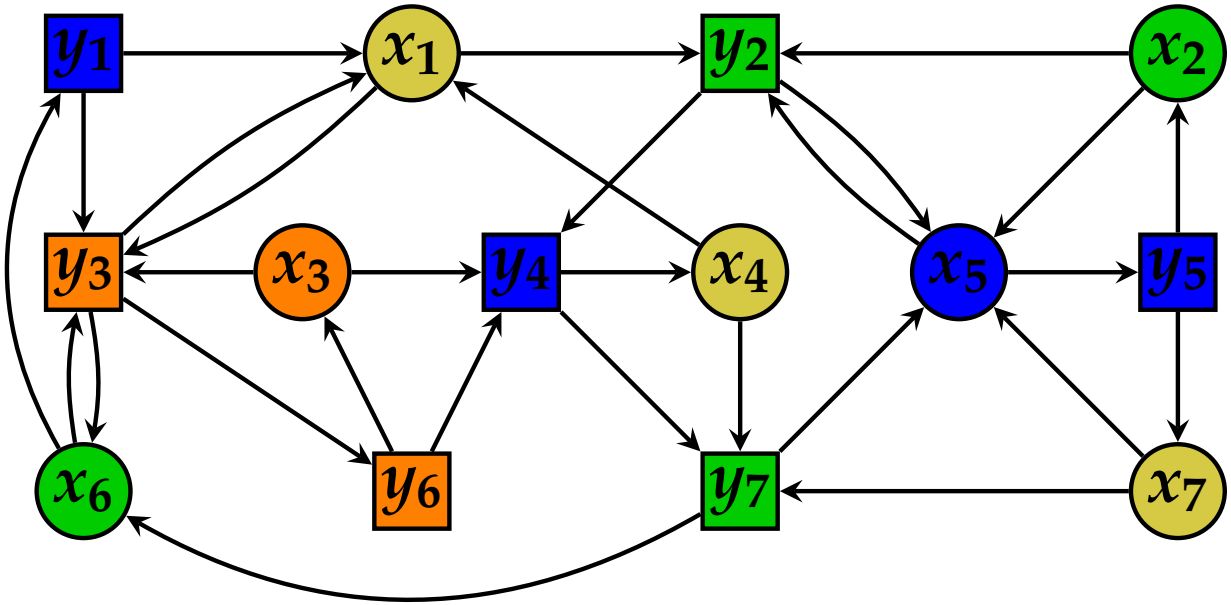
A play in G is an infinite sequence $\alpha = v_0 v_1 v_2 \cdots$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$.

By $c(\alpha)$ we denote the corresponding sequence of colors $c(v_0)c(v_1)c(v_2)\cdots$

Plays

A play in G is an infinite sequence $\alpha = v_0 v_1 v_2 \cdots$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$.

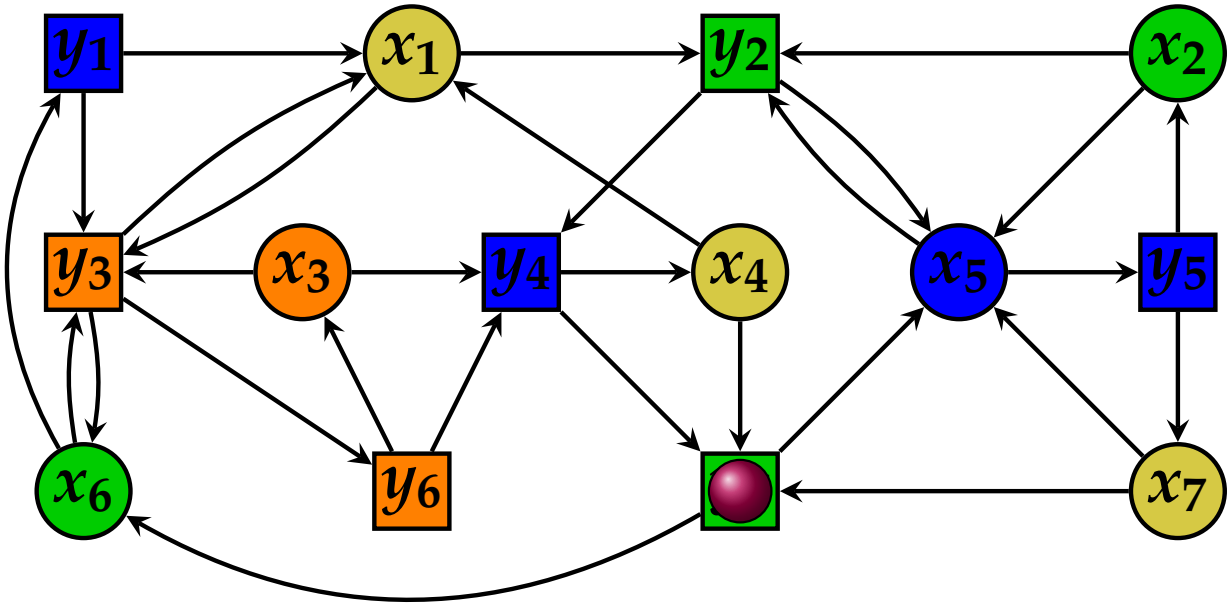
By $c(\alpha)$ we denote the corresponding sequence of colors $c(v_0)c(v_1)c(v_2)\cdots$



Plays

A play in G is an infinite sequence $\alpha = v_0 v_1 v_2 \cdots$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$.

By $c(\alpha)$ we denote the corresponding sequence of colors $c(v_0)c(v_1)c(v_2)\cdots$



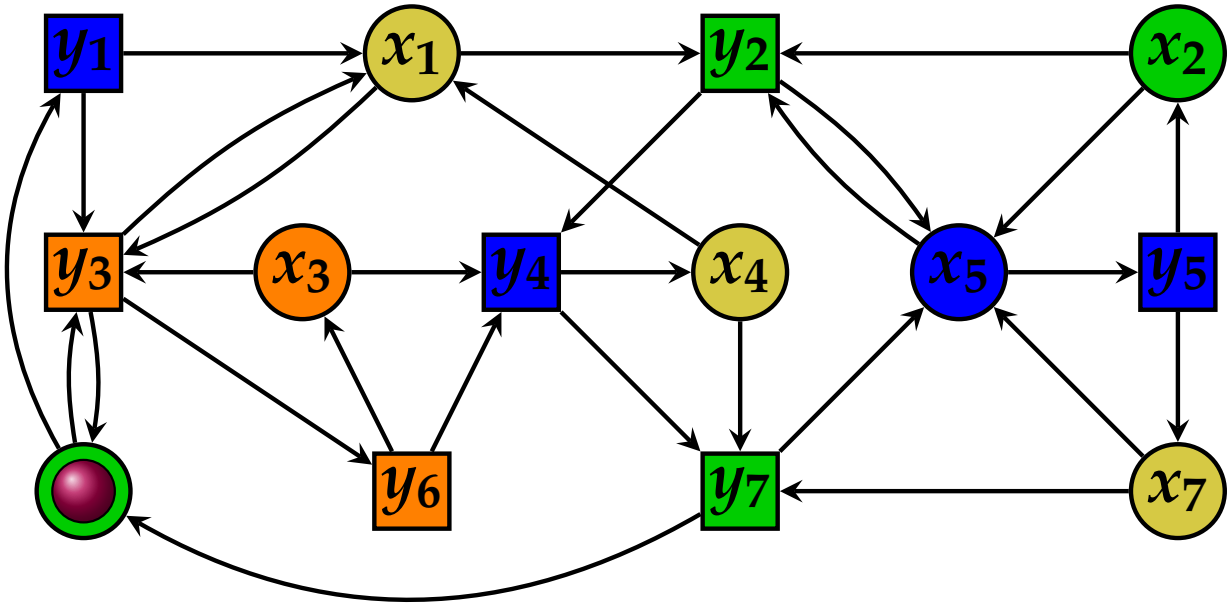
α : y_7

$c(\alpha)$: ●

Plays

A play in G is an infinite sequence $\alpha = v_0 v_1 v_2 \cdots$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$.

By $c(\alpha)$ we denote the corresponding sequence of colors $c(v_0)c(v_1)c(v_2)\cdots$



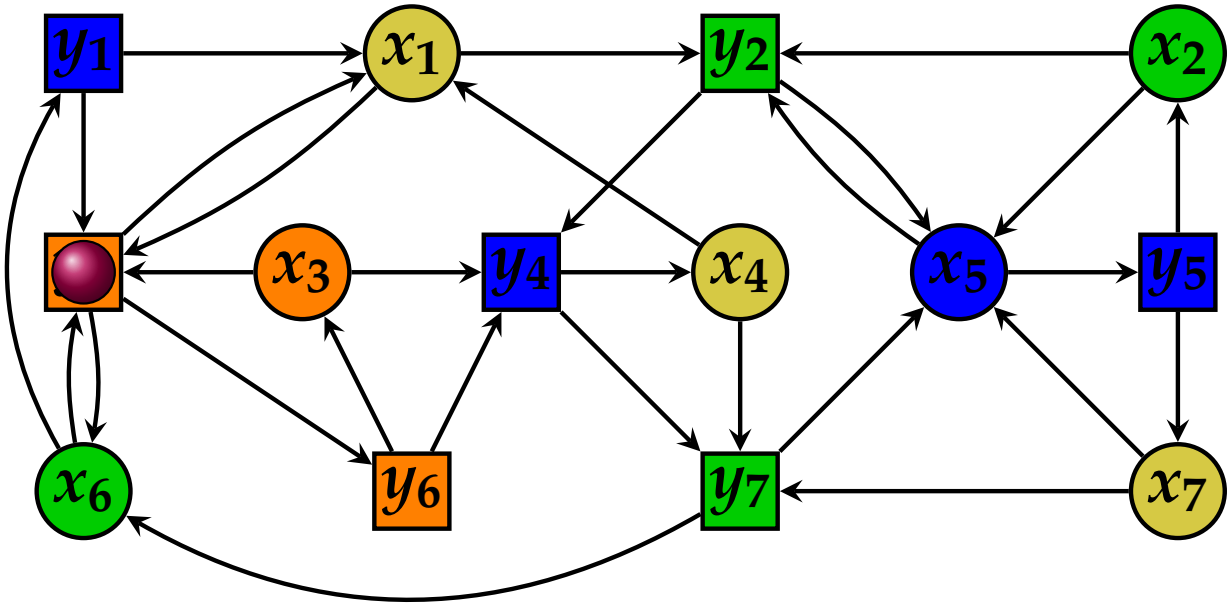
α : y_7 x_6

$c(\alpha)$: ● ●

Plays

A play in G is an infinite sequence $\alpha = v_0 v_1 v_2 \cdots$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$.

By $c(\alpha)$ we denote the corresponding sequence of colors $c(v_0)c(v_1)c(v_2)\cdots$



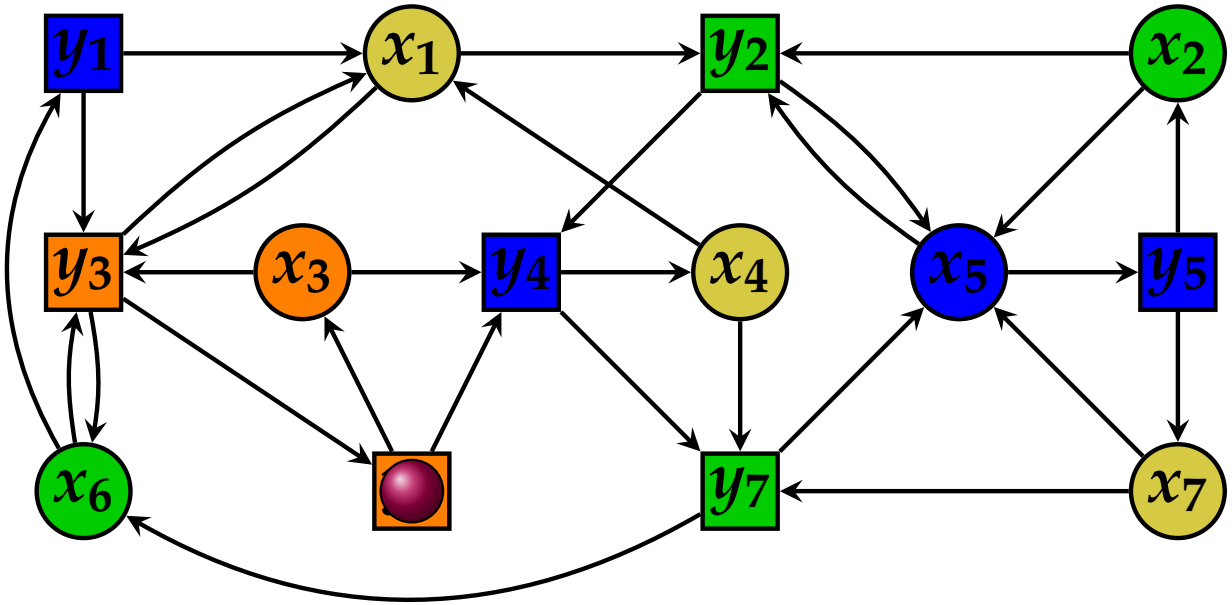
α : y_7 x_6 y_3

$c(\alpha)$: ● ● ●

Plays

A play in G is an infinite sequence $\alpha = v_0 v_1 v_2 \cdots$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$.

By $c(\alpha)$ we denote the corresponding sequence of colors $c(v_0)c(v_1)c(v_2)\cdots$



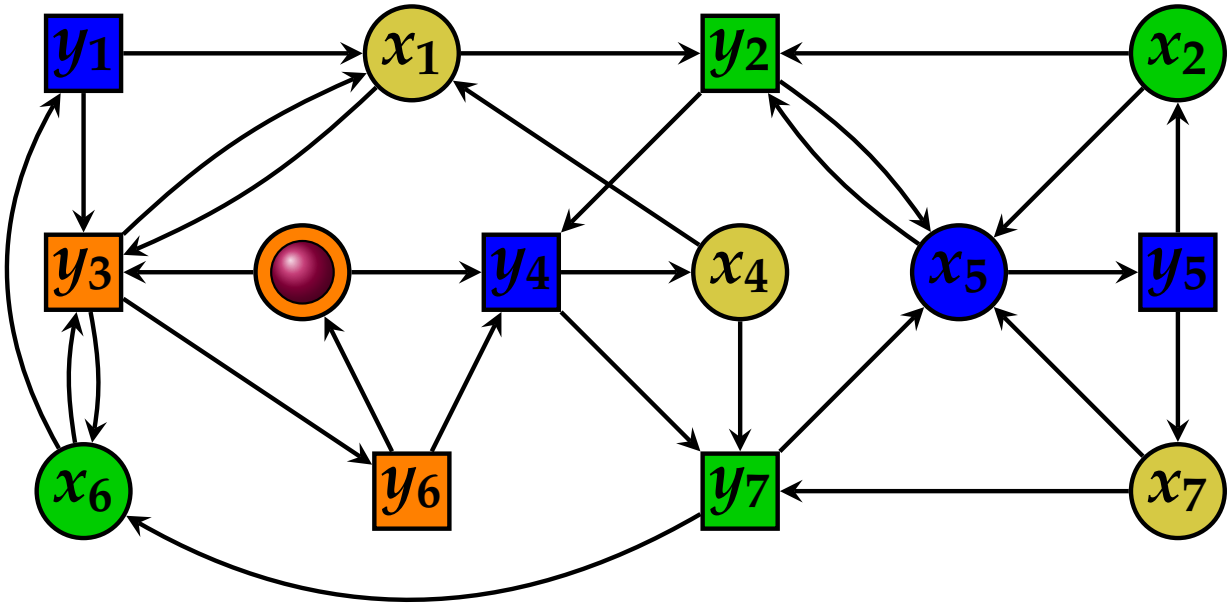
α : y_7 x_6 y_3 y_6

$c(\alpha)$: ● ● ● ●

Plays

A play in G is an infinite sequence $\alpha = v_0 v_1 v_2 \cdots$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$.

By $c(\alpha)$ we denote the corresponding sequence of colors $c(v_0)c(v_1)c(v_2)\cdots$



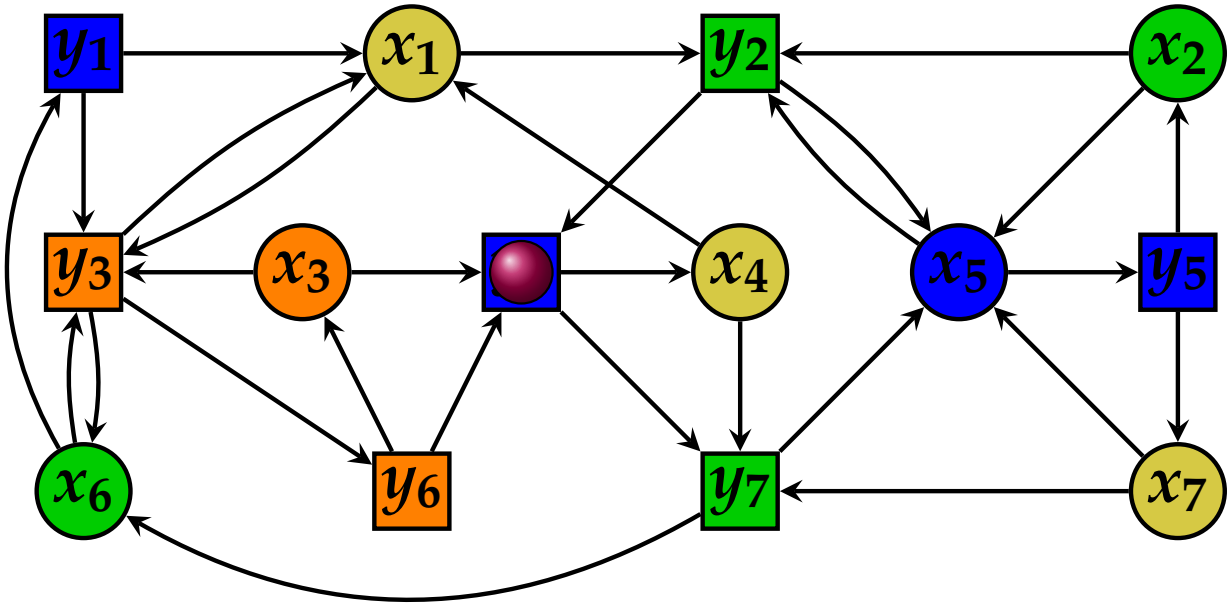
α : y_7 x_6 y_3 y_6 x_3

$c(\alpha)$: ● ● ● ● ●

Plays

A play in G is an infinite sequence $\alpha = v_0 v_1 v_2 \cdots$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$.

By $c(\alpha)$ we denote the corresponding sequence of colors $c(v_0)c(v_1)c(v_2)\cdots$



$$\begin{array}{lccccccc} \alpha: & y_7 & x_6 & y_3 & y_6 & x_3 & y_4 & \cdots \\ c(\alpha): & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \cdots \end{array}$$

Game

$\mathcal{G} = (G, Win)$

- G: game graph
- $Win \subseteq C^\omega$: winning condition

Eva wins a play α if $c(\alpha) \in Win$. Otherwise Adam wins.

Examples for winning conditions:

- plays that contain a green vertex somewhere
- plays that infinitely often contain a blue vertex
- plays that contain infinitely often a blue vertex and finitely often a yellow vertex, or infinitely often a green vertex
- $\ldots$

Remarks

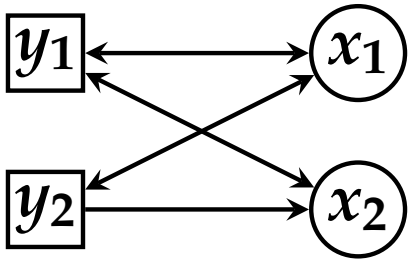
Assumptions on the graphs:

- **Game graphs can be countably infinite**
- **All vertices v have at least one successor: $E(v) \neq \emptyset$**
- **For simplicity we assume finite branching**

Concerning the coloring, we are flexible:

- **We can also use numbers, letters, etc. as colors in C**
- **Often we identify C and V and specify Win directly using the names of the vertices**

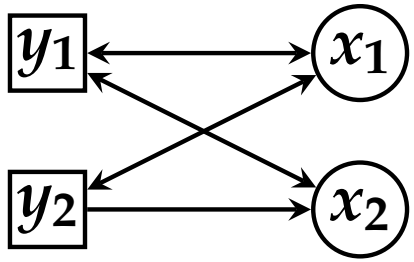
Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Trying to Win – Strategies



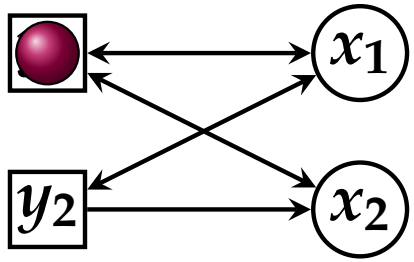
Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

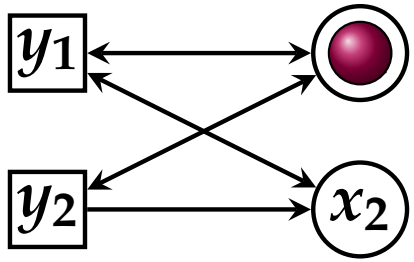
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

$$\alpha: \quad y_1$$

Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

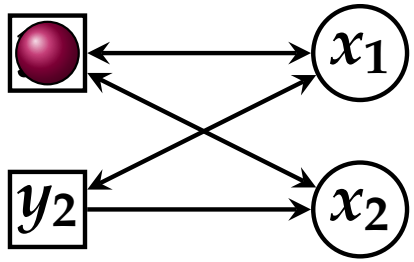
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

$$\alpha: \quad y_1 \quad x_1$$

Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

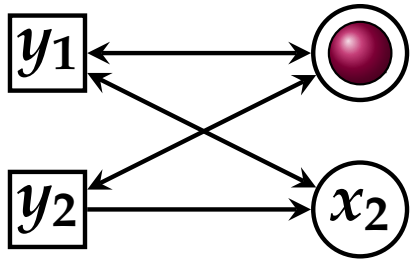
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

$$\alpha: \quad y_1 \quad x_1 \quad y_1$$

Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

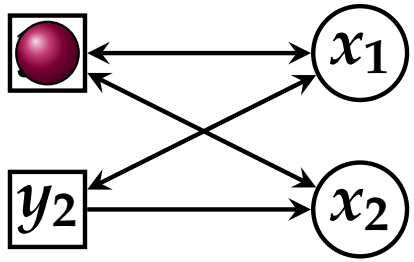
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

$$\alpha: \quad y_1 \quad x_1 \quad y_1 \quad x_1$$

Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

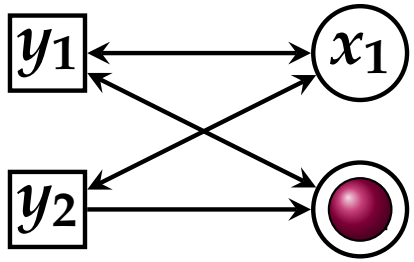
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

$$\alpha: \quad y_1 \quad x_1 \quad y_1 \quad x_1 \quad y_1$$

Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

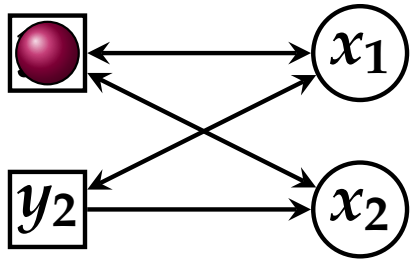
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

$$\alpha: \quad y_1 \quad x_1 \quad y_1 \quad x_1 \quad y_1 \quad x_2$$

Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

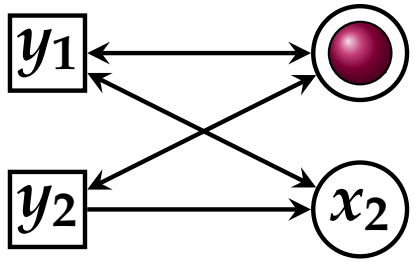
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

$$\alpha: \quad y_1 \quad x_1 \quad y_1 \quad x_1 \quad y_1 \quad x_2 \quad y_1$$

Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

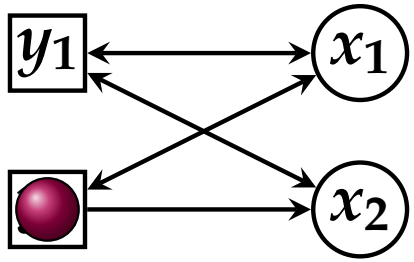
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

$$\alpha: \quad y_1 \quad x_1 \quad y_1 \quad x_1 \quad y_1 \quad x_2 \quad y_1 \quad x_1$$

Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

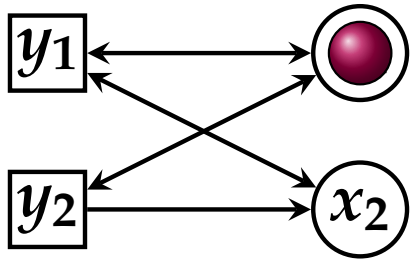
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

$$\alpha: \quad y_1 \quad x_1 \quad y_1 \quad x_1 \quad y_1 \quad x_2 \quad y_1 \quad x_1 \quad y_2$$

Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

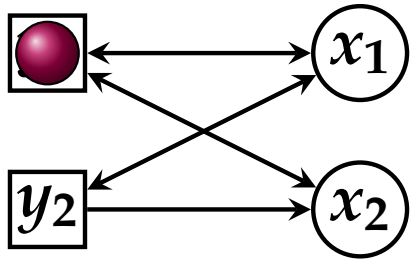
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

$$\alpha: \quad y_1 \quad x_1 \quad y_1 \quad x_1 \quad y_1 \quad x_2 \quad y_1 \quad x_1 \quad y_2 \quad x_1$$

Trying to Win – Strategies



Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Eva can win by playing as follows when the play is at x_1:

- **If the previous x_i was x_2: move to y_2**
- **Otherwise: move to y_1**

$$\alpha: \quad y_1 \quad x_1 \quad y_1 \quad x_1 \quad y_1 \quad x_2 \quad y_1 \quad x_1 \quad y_2 \quad x_1 \quad y_1 \quad \cdots$$

Strategies – Formal

- **A strategy for Eva is a function**

$$f : V^* V_\exists \longrightarrow V$$

with $f(xv) = v'$ **implies** $(v, v') \in E$

Strategies – Formal

- **A strategy for Eva is a function**

$$f : V^* V_\exists \to V$$

  with $f(xv) = v'$ **implies** $(v, v') \in E$

- **A play** $v_0 v_1 v_2 \cdots$ **is played according to** f **if**

$$\forall i : v_i \in V_\exists \to f(v_0 \cdots v_i) = v_{i+1}$$

- $Out(f, v_0) =$ **set of all plays starting in** v_0 **that are played according to** f **(the possible outcomes of** f**).**

Strategies – Formal

- **A strategy for Eva is a function**

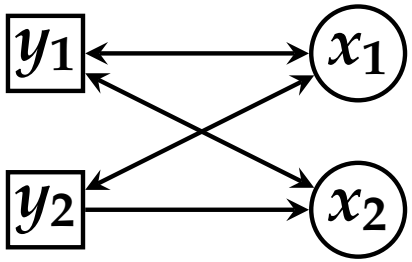
$$f : V^* V_\exists \to V$$

  with $f(xv) = v'$ implies $(v, v') \in E$
- **A play $v_0 v_1 v_2 \cdots$ is played according to f if**

$$\forall i : v_i \in V_\exists \to f(v_0 \cdots v_i) = v_{i+1}$$

- **$Out(f, v_0) =$ set of all plays starting in v_0 that are played according to f (the possible outcomes of f).**
- **A strategy for Eva is a winning strategy from vertex v_0 if $Out(f, v_0) \subseteq Win$**
- **A strategy for Adam (defined similarly) is a winning strategy from vertex v_0 if $Out(f, v_0) \cap Win = \emptyset$**

Memory for Strategies

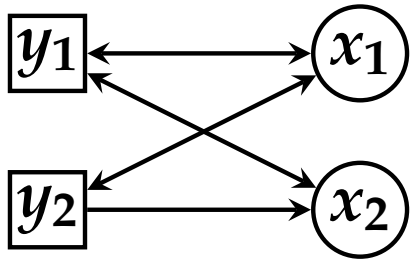


$$y_2 \in \mathbf{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathbf{Inf}(\alpha)$$

Possible strategies for Eva (at x_1):

1. **Move to y_2 iff previous x_i was x_2.**
2. **Move to y_2 iff there have been more visits to x_2 than to y_2.**

Memory for Strategies



$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

Possible strategies for Eva (at x_1):

1. **Move to y_2 iff previous x_i was x_2.**
2. **Move to y_2 iff there have been more visits to x_2 than to y_2.**

The first strategy only requires two states of memory, the second one infinitely many.

Winning Areas

The winning area for Eva is the set $W_\exists$ of all vertices from which Eva has a winning strategy:

$$W_\exists = \{v \in V \mid \textbf{Eva has a winning strategy from } v\}.$$

The winning area for Adam is denoted $W_\forall$ and is defined in the same way.

Determinacy

A game $\mathcal{G} = (G, Win)$ is determined if from each node either Eva or Adam has a winning strategy, i.e., if

$$W_\exists \cup W_\forall = V$$

Determinacy

A game $\mathcal{G} = (G, Win)$ is determined if from each node either Eva or Adam has a winning strategy, i.e., if

$$W_\exists \cup W_\forall = V$$

Determinacy is about swapping quantifiers

Eva does not have a winning strategy:

$$\forall f_\exists \exists f_\forall \; (\; f_\forall \text{ wins against } f_\exists)$$

Determinacy

A game $\mathcal{G} = (G, Win)$ is determined if from each node either Eva or Adam has a winning strategy, i.e., if

$$W_\exists \cup W_\forall = V$$

Determinacy is about swapping quantifiers

Eva does not have a winning strategy:

$$\forall f_\exists \exists f_\forall \; (\; f_\forall \text{ wins against } f_\exists)$$

By determinacy:

$$\exists f_\forall \forall f_\exists \; (\; f_\forall \text{ wins against } f_\exists)$$

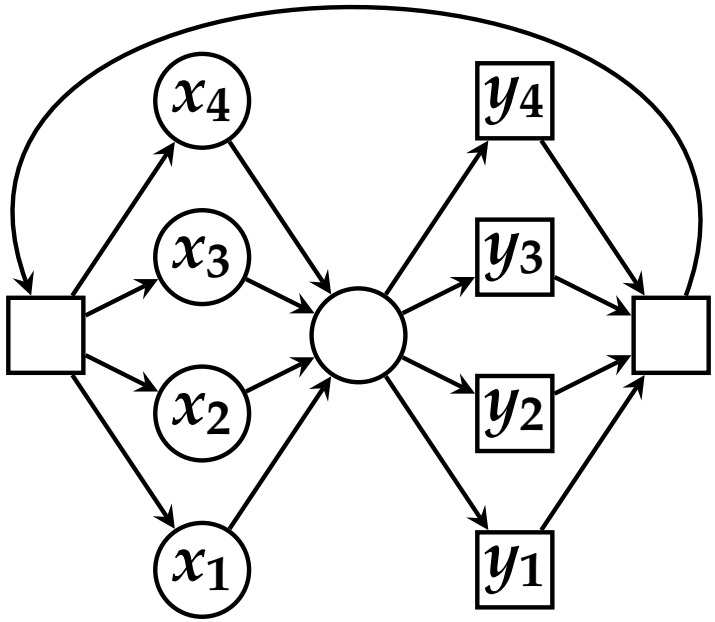
Central Problems

- **Determinacy**
- **Computing winning areas and strategies**
- **Memory required for strategies**

Central Problems

- **Determinacy**
- **Computing winning areas and strategies**
- **Memory required for strategies**

A difficult game:



$$\alpha \in \mathit{Win} \text{ iff } \max\{i \mid y_i \in \mathrm{Inf}(\alpha)\} = |\{i \mid x_i \in \mathrm{Inf}(\alpha)\}|$$

Summary

- **Games: game graph (arena) and winning condition**
- **Strategies** $f : V^* V_\exists \longrightarrow V$
- **Winning areas** $W_\exists, W_\forall$
- **Determinacy** $W_\exists \cup W_\forall = V$

1 Basics

2 Positional Strategies: Reachability and Attractors

3 Büchi Games

4 Muller Games and Finite Memory Strategies

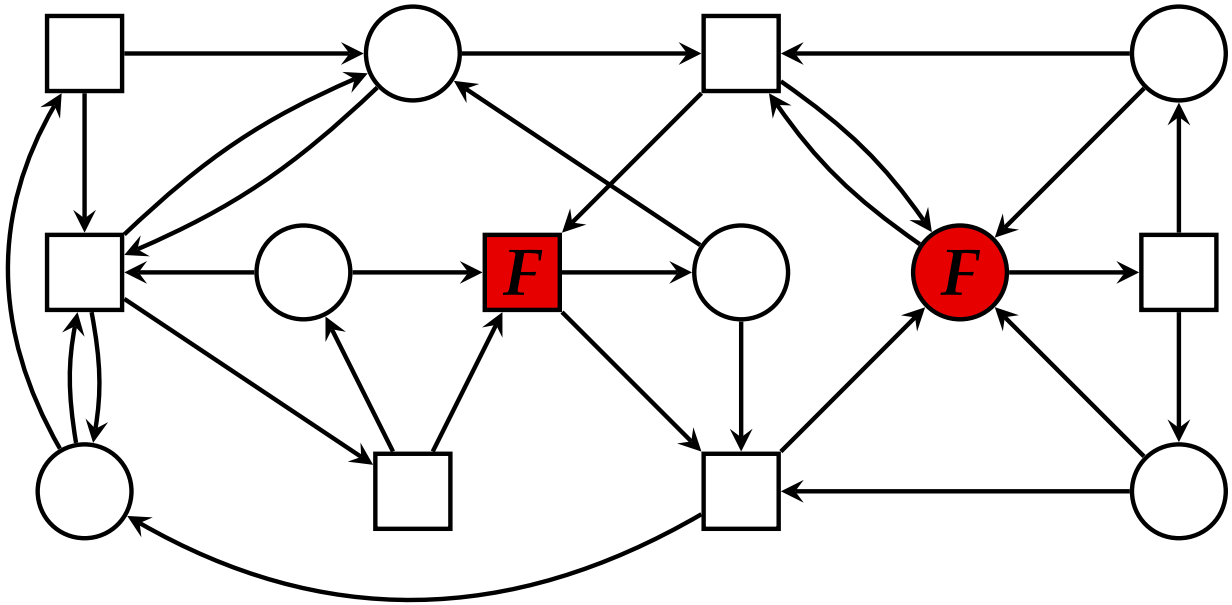
5 Parity Games

- Positional Determinacy
- Algorithms for Parity Games
- Reducing Muller Games to Parity Games

6 Applications

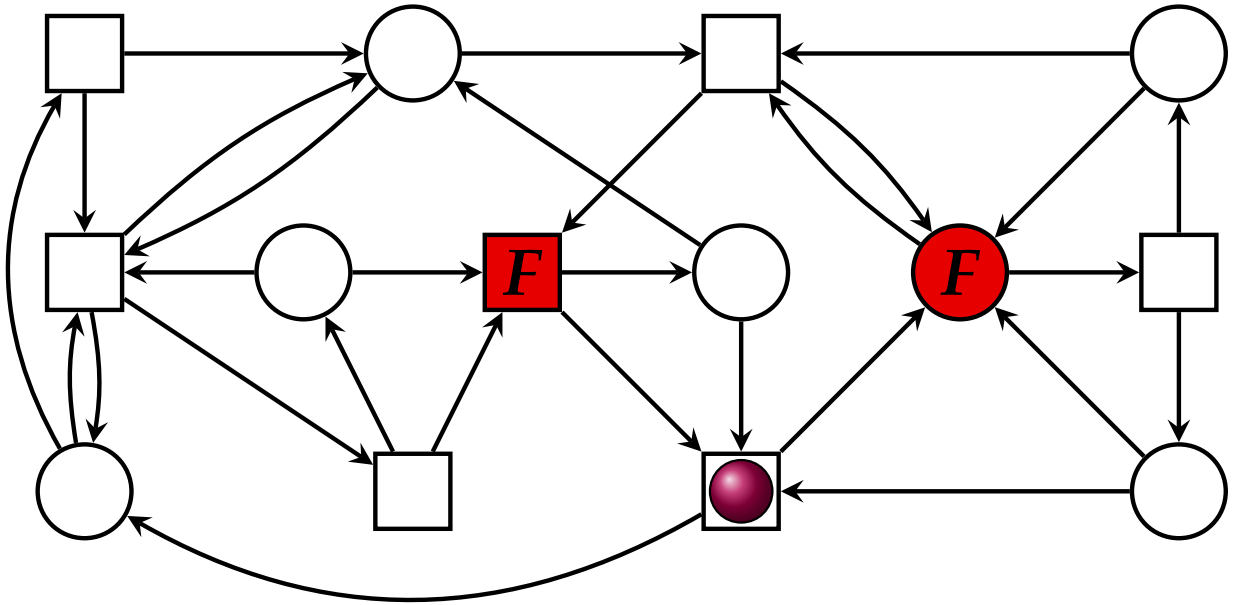
Reachability Condition

- A winning condition $Win \subseteq C^\omega$ is called a reachability condition if there is $D \subseteq C$ such that $\alpha \in Win$ iff some color from D occurs in α.
- The objective for Eva in a reachability game is to reach some vertex with color D.
- Normally, we specify a reachability game as $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



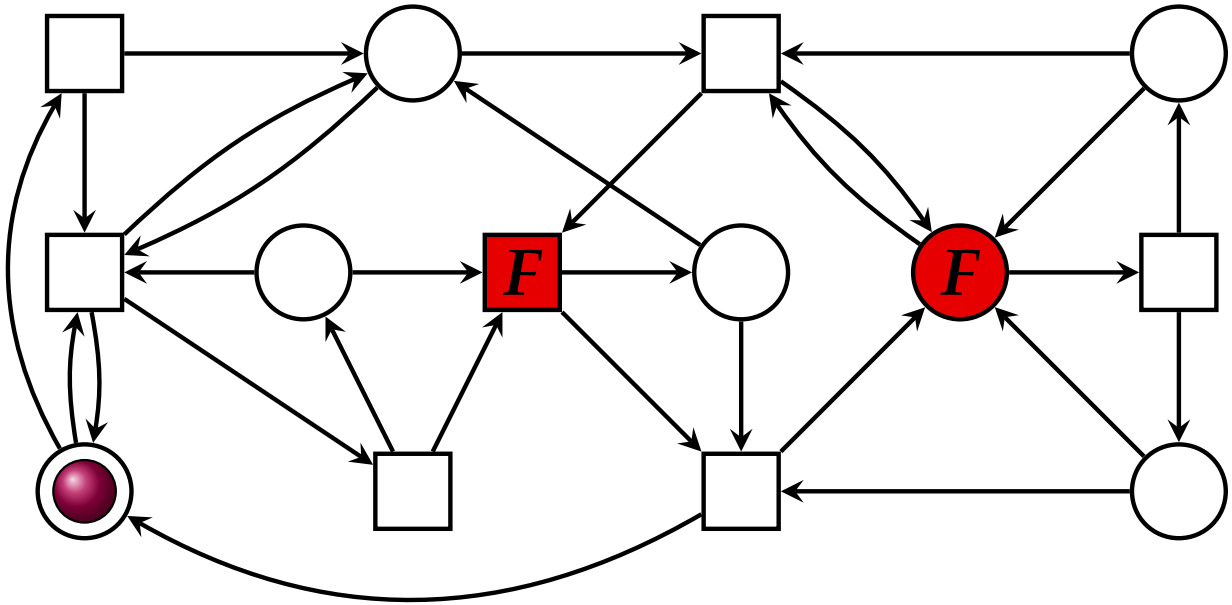
Reachability Condition

- A winning condition $Win \subseteq C^\omega$ is called a reachability condition if there is $D \subseteq C$ such that $\alpha \in Win$ iff some color from D occurs in α.
- The objective for Eva in a reachability game is to reach some vertex with color D.
- Normally, we specify a reachability game as $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



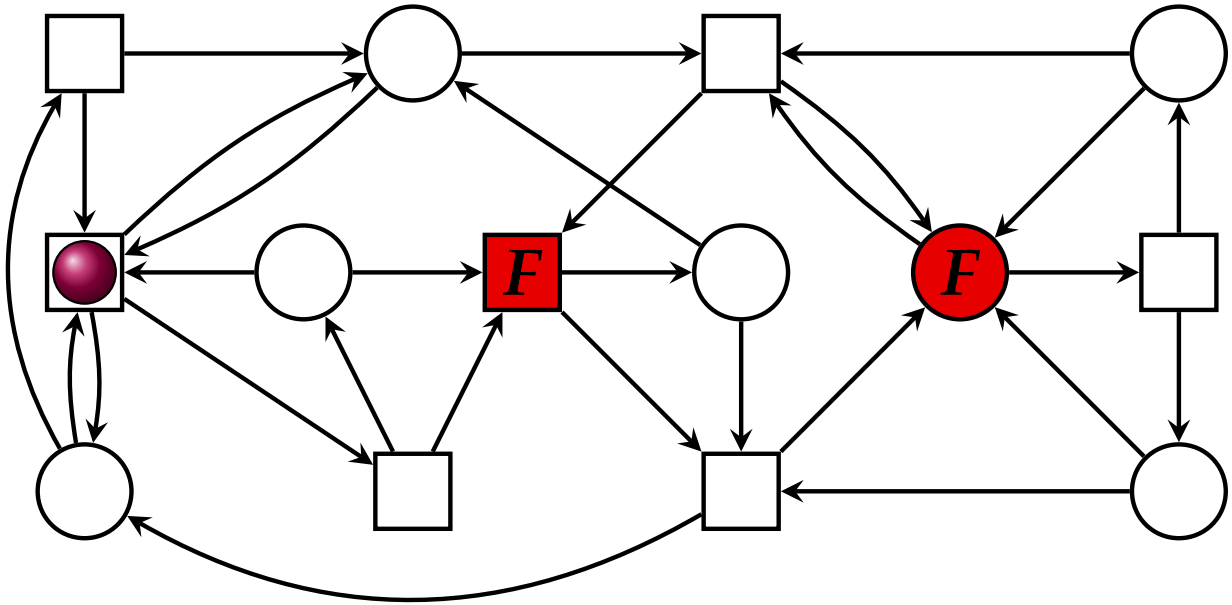
Reachability Condition

- A winning condition $Win \subseteq C^\omega$ is called a reachability condition if there is $D \subseteq C$ such that $\alpha \in Win$ iff some color from D occurs in α.
- The objective for Eva in a reachability game is to reach some vertex with color D.
- Normally, we specify a reachability game as $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



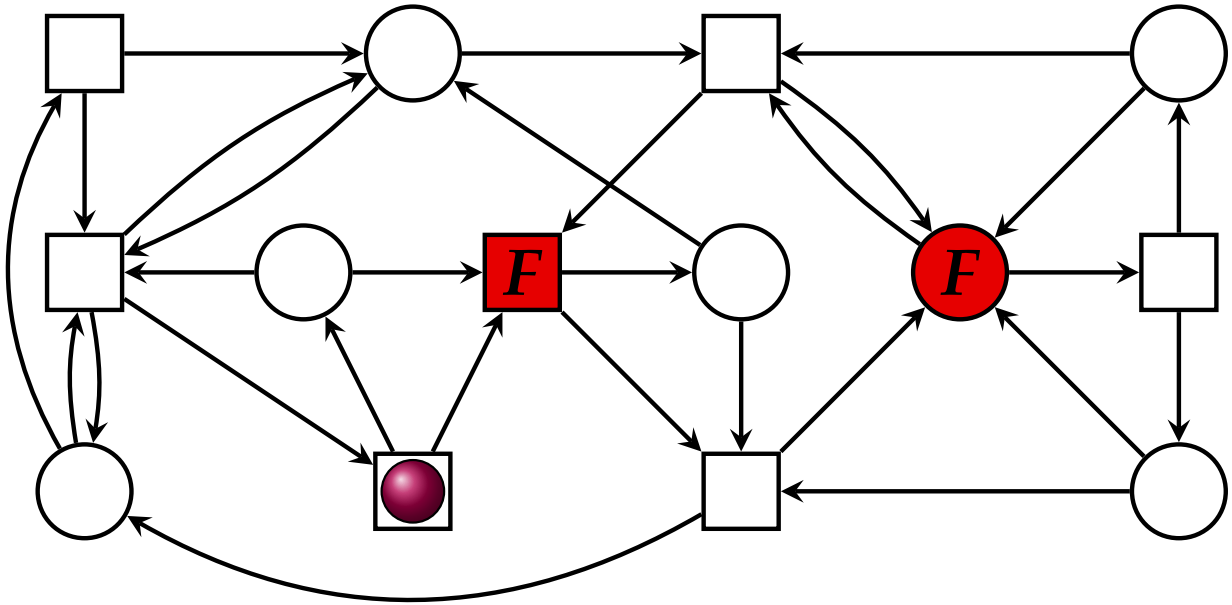
Reachability Condition

- A winning condition $Win \subseteq C^\omega$ is called a reachability condition if there is $D \subseteq C$ such that $\alpha \in Win$ iff some color from D occurs in α.
- The objective for Eva in a reachability game is to reach some vertex with color D.
- Normally, we specify a reachability game as $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



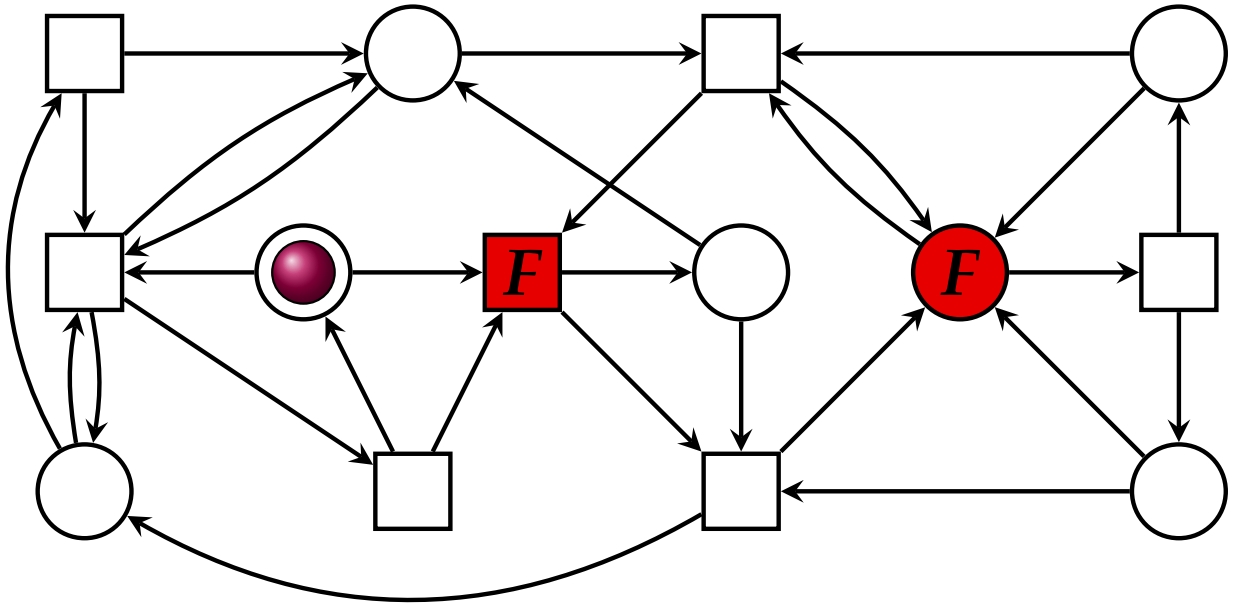
Reachability Condition

- A winning condition $Win \subseteq C^\omega$ is called a reachability condition if there is $D \subseteq C$ such that $\alpha \in Win$ iff some color from D occurs in α.
- The objective for Eva in a reachability game is to reach some vertex with color D.
- Normally, we specify a reachability game as $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



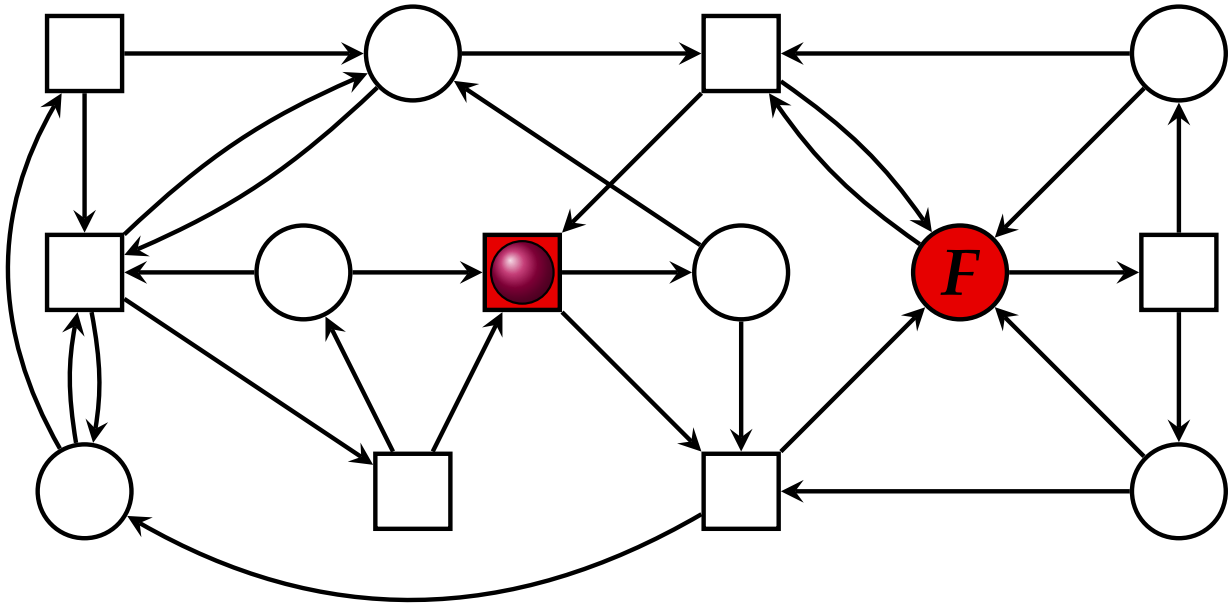
Reachability Condition

- A winning condition $Win \subseteq C^\omega$ is called a reachability condition if there is $D \subseteq C$ such that $\alpha \in Win$ iff some color from D occurs in α.
- The objective for Eva in a reachability game is to reach some vertex with color D.
- Normally, we specify a reachability game as $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



Reachability Condition

- A winning condition $Win \subseteq C^\omega$ is called a reachability condition if there is $D \subseteq C$ such that $\alpha \in Win$ iff some color from D occurs in α.
- The objective for Eva in a reachability game is to reach some vertex with color D.
- Normally, we specify a reachability game as $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



Operator Pre

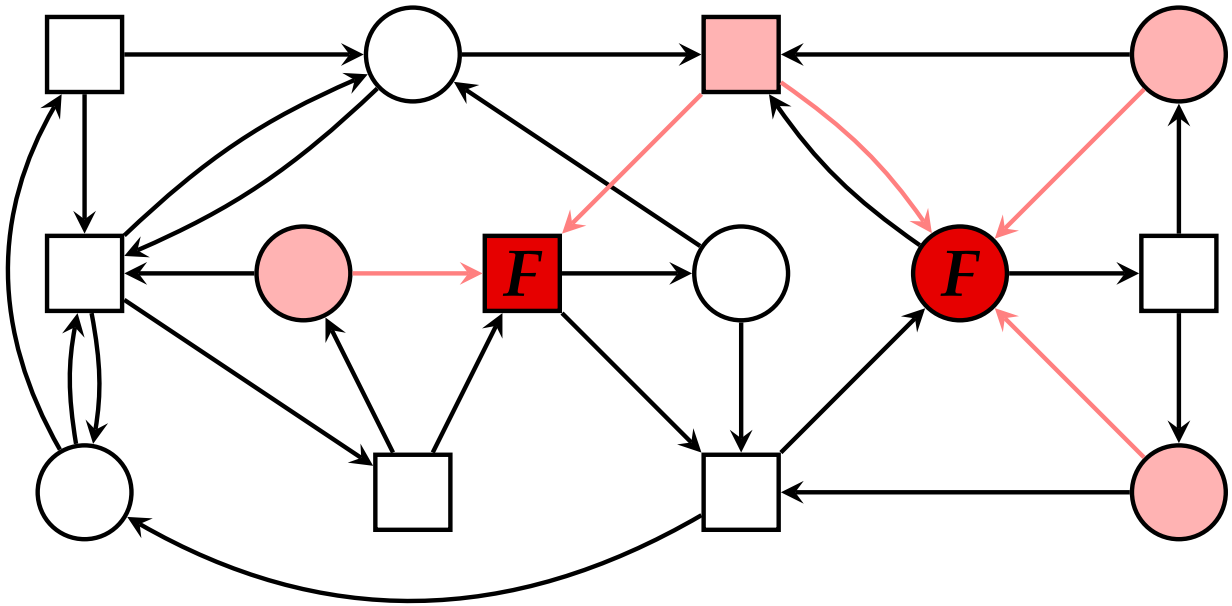
On $Pre_{\exists}(F)$ Eva can force to reach F in one step:

$$\begin{aligned} Pre_{\exists}(F) &= \{v \in V_{\exists} \mid \exists v' \in E(v) \cap F\} \\ &\quad \cup \{v \in V_{\forall} \mid \neg\exists v' \in E(v) \setminus F\} \end{aligned}$$

Operator Pre

On $Pre_\exists(F)$ Eva can force to reach F in one step:

$$\begin{aligned} Pre_\exists(F) &= \{v \in V_\exists \mid \exists v' \in E(v) \cap F\} \\ &\quad \cup \{v \in V_\forall \mid \neg\exists v' \in E(v) \setminus F\} \end{aligned}$$

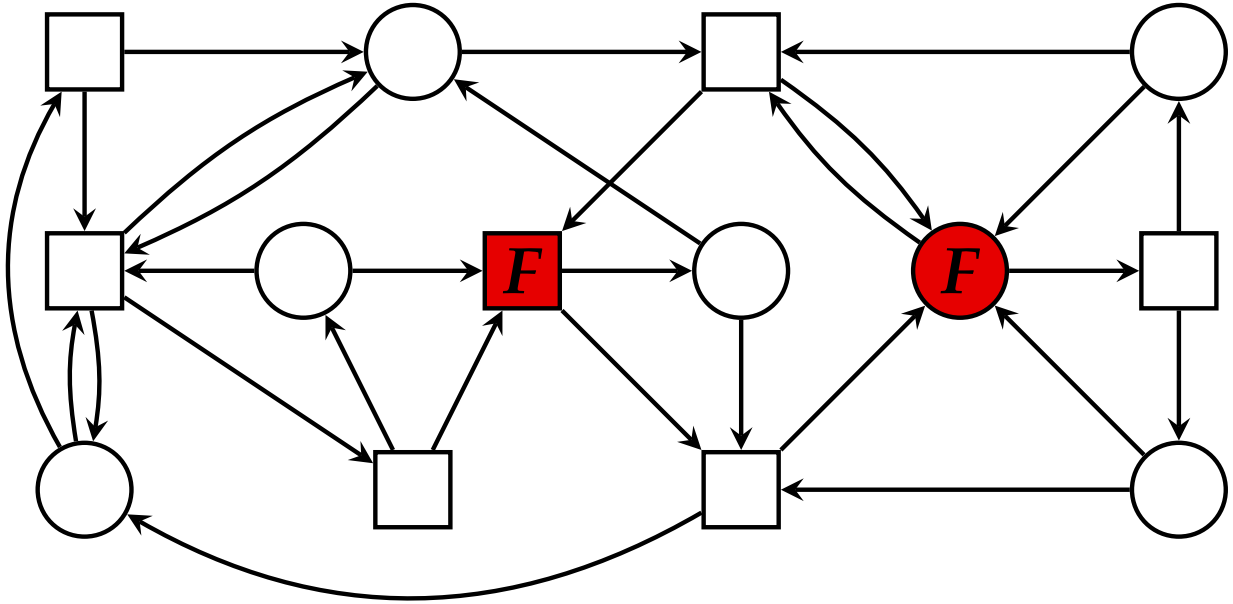


Attractor

The attractor for Eva of a set F is the set of all nodes from which Eva can force to reach F in finitely many moves:

$$Attr_{\exists}(F) = \bigcup_{i \geq 0} Attr^{i}_{\exists}(F)$$

- $Attr^{0}_{\exists}(F) = F$
- $Attr^{i+1}_{\exists}(F) = Attr^{i}_{\exists}(F) \cup Pre_{\exists}(Attr^{i}_{\exists}(F))$

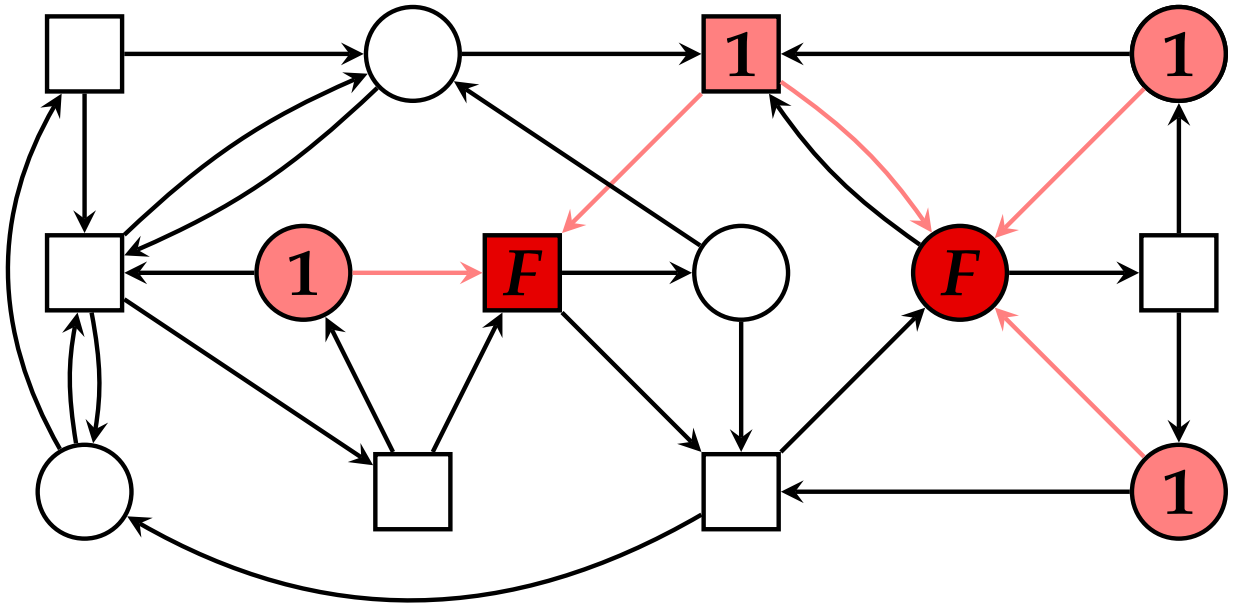


Attractor

The attractor for Eva of a set F is the set of all nodes from which Eva can force to reach F in finitely many moves:

$$Attr_{\exists}(F) = \bigcup_{i \geq 0} Attr^{i}_{\exists}(F)$$

- $Attr^{0}_{\exists}(F) = F$
- $Attr^{i+1}_{\exists}(F) = Attr^{i}_{\exists}(F) \cup Pre_{\exists}(Attr^{i}_{\exists}(F))$

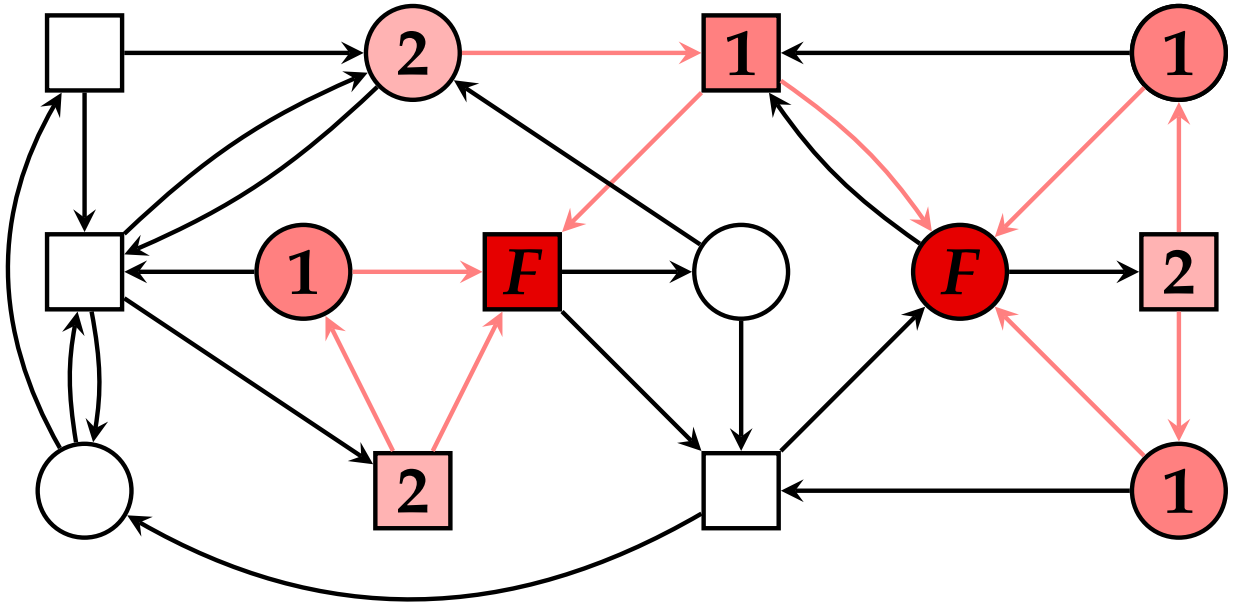


Attractor

The attractor for Eva of a set F is the set of all nodes from which Eva can force to reach F in finitely many moves:

$$Attr_{\exists}(F) = \bigcup_{i \geq 0} Attr^{i}_{\exists}(F)$$

- $Attr^{0}_{\exists}(F) = F$
- $Attr^{i+1}_{\exists}(F) = Attr^{i}_{\exists}(F) \cup Pre_{\exists}(Attr^{i}_{\exists}(F))$

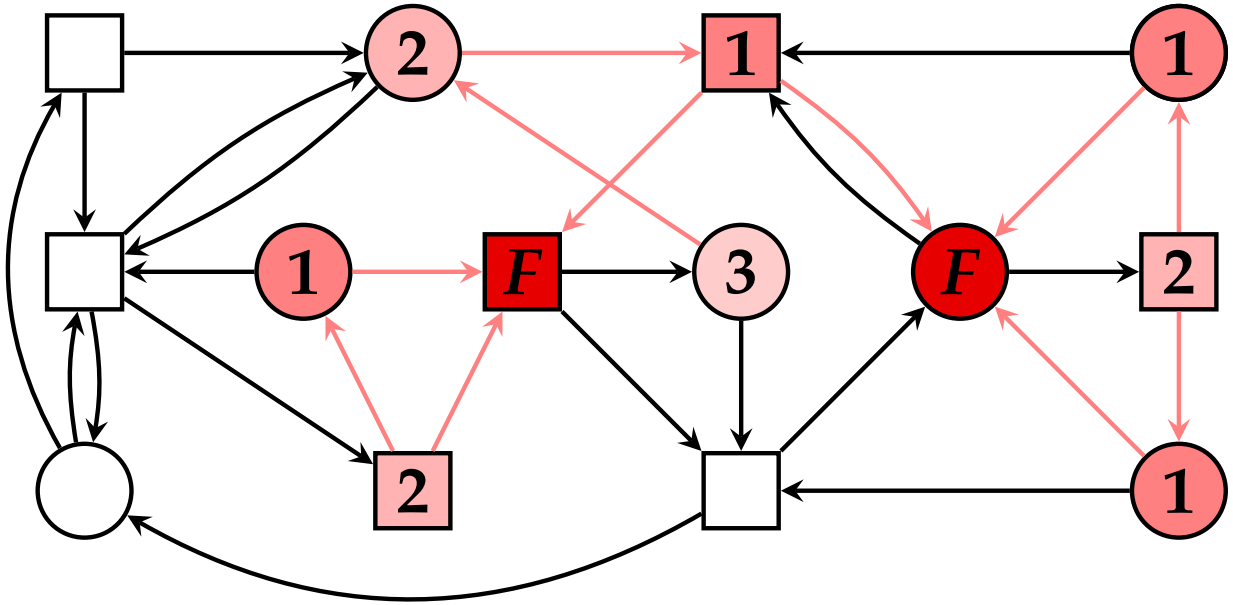


Attractor

The attractor for Eva of a set F is the set of all nodes from which Eva can force to reach F in finitely many moves:

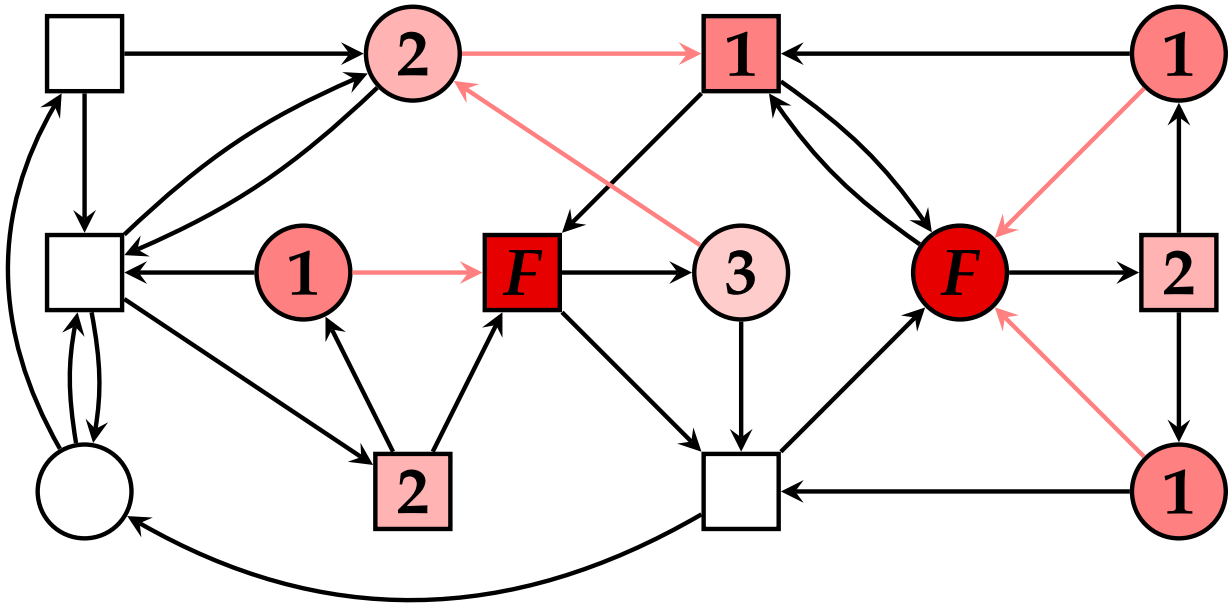
$$Attr_{\exists}(F) = \bigcup_{i \geq 0} Attr^{i}_{\exists}(F)$$

- $Attr^{0}_{\exists}(F) = F$
- $Attr^{i+1}_{\exists}(F) = Attr^{i}_{\exists}(F) \cup Pre_{\exists}(Attr^{i}_{\exists}(F))$



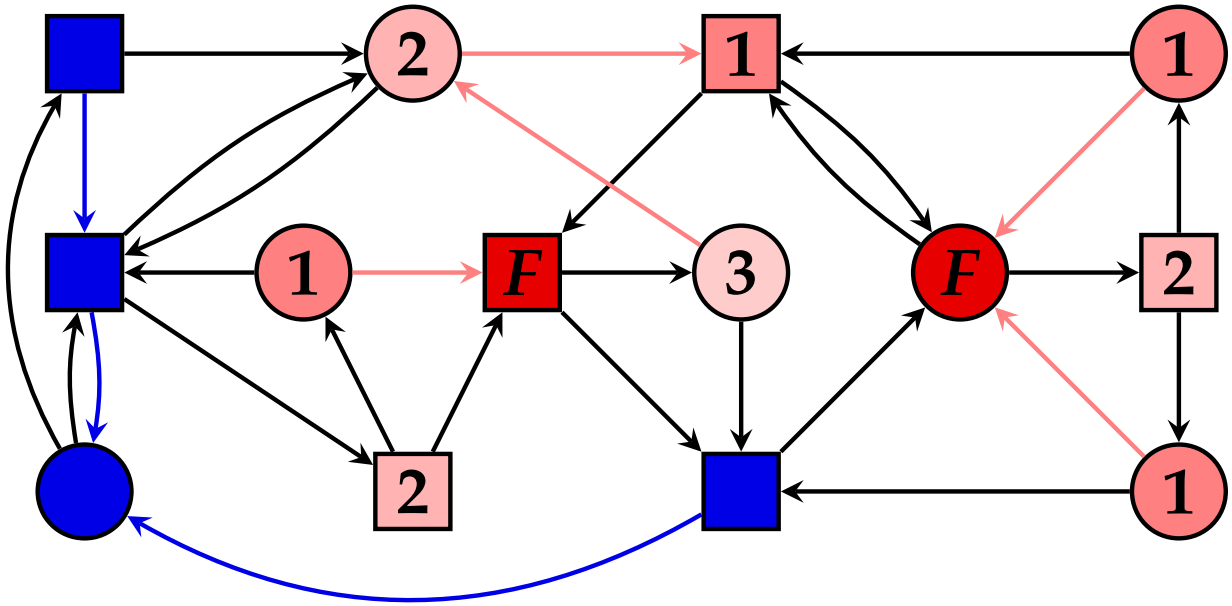
Attractor and Trap Strategy

- On $Attr^i_\exists(F)$ Eva can ensure that the next move leads to $Attr^j_\exists(F)$ for $j < i$. Call this an attractor strategy.



Attractor and Trap Strategy

- On $Attr^i_\exists(F)$ Eva can ensure that the next move leads to $Attr^j_\exists(F)$ for $j < i$. Call this an attractor strategy.
- The complement $V \setminus Attr_\exists(F)$ is a trap for Eva: Adam can ensure that the play remains in this set. Call this a trap strategy for Adam.



Positional Strategies

- A positional strategy for Eva is a mapping

$$f : V_\exists \to V$$

  with $(v, f(v)) \in E$ for all $v \in V_\exists$. Similarly for Adam.
- Positional strategies are also called memoryless.
- We call a game positionally determined if from each vertex one of the players has a positional winning strategy.
- Positional strategies can be defined by selecting one outgoing edge for each vertex of the player.

Determinacy of Reachability Games

Theorem

Reachability games $\mathcal{G} = (G, F)$ are positionally determined with $W_\exists = Attr_\exists(F)$.

Determinacy of Reachability Games

Theorem

Reachability games $\mathcal{G} = (G, F)$ are positionally determined with $W_\exists = Attr_\exists(F)$.

This theorem was already shown in 1913 by E. Zermelo:

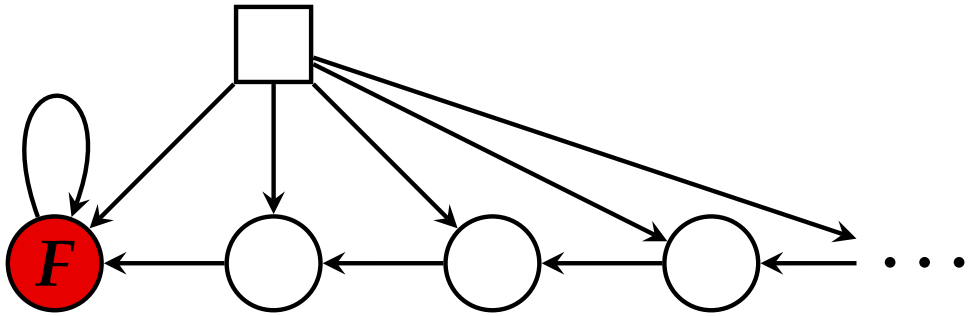
“Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels”
(On an application of set theory to the theory of chess)

Final Remarks

- **An attractor can be computed in linear time (in the size of the graph)**

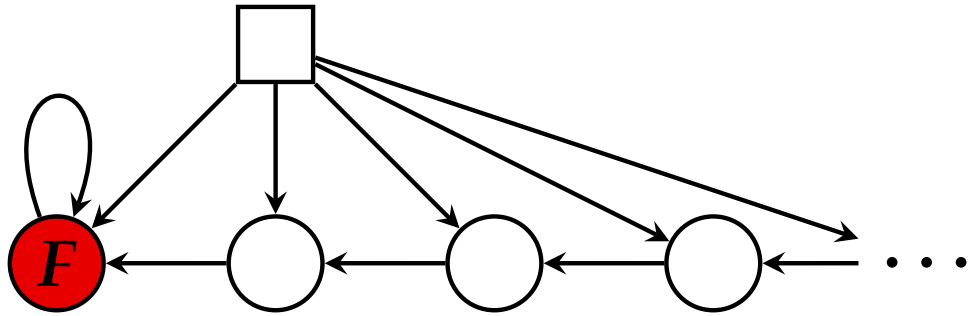
Final Remarks

- **An attractor can be computed in linear time (in the size of the graph)**
- **For graphs with infinite degree it is not enough to stop the attractor computation after ω iterations.**
  Illustration:



Final Remarks

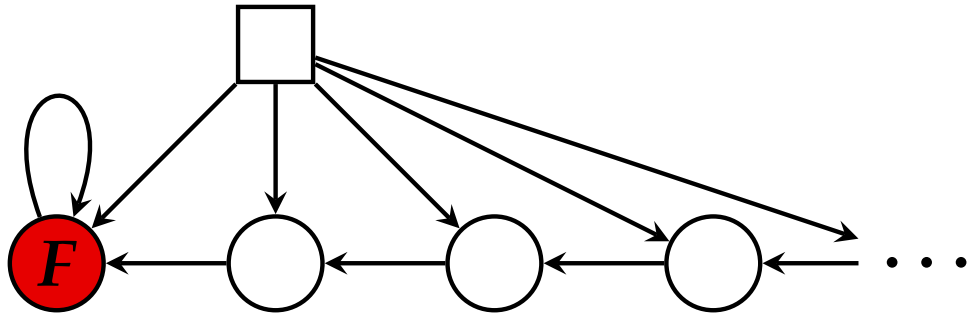
- **An attractor can be computed in linear time (in the size of the graph)**
- **For graphs with infinite degree it is not enough to stop the attractor computation after ω iterations.**
  Illustration:



- **A trap defines a subgame (each vertex has at least one edge staying inside the trap). This is important when inductively solving games.**

Final Remarks

- An attractor can be computed in linear time (in the size of the graph)
- For graphs with infinite degree it is not enough to stop the attractor computation after ω iterations.
  Illustration:



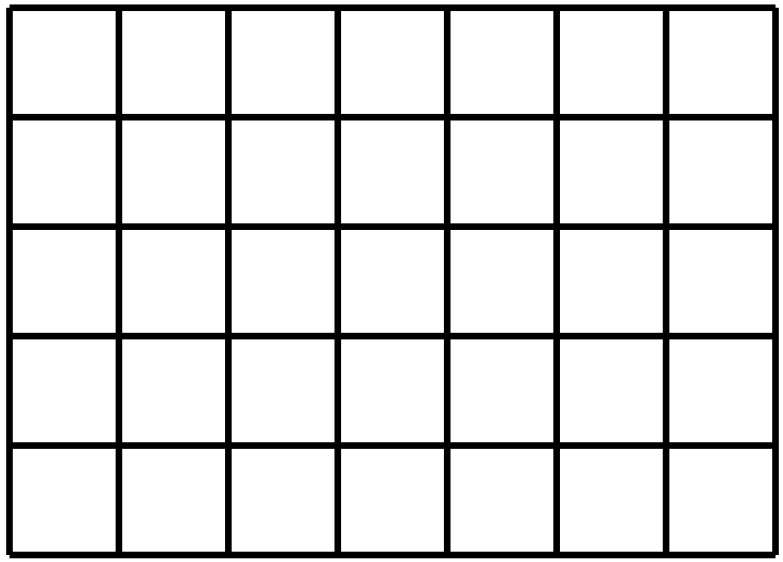
- A trap defines a subgame (each vertex has at least one edge staying inside the trap). This is important when inductively solving games.
- The dual condition of avoiding a set of vertices is called safety condition. A reachability condition is a safety condition for the opponent.

Summary

- **Reachability games: goal of Eva is to reach a set F of vertices**
- **Attractor $Attr_{\exists}(F)$ (based on $Pre_{\exists}$)**
- **Complement of an attractor is a trap for the other player**
- **Attractor and trap strategies are positional**
- **Reachability games are positionally determined**

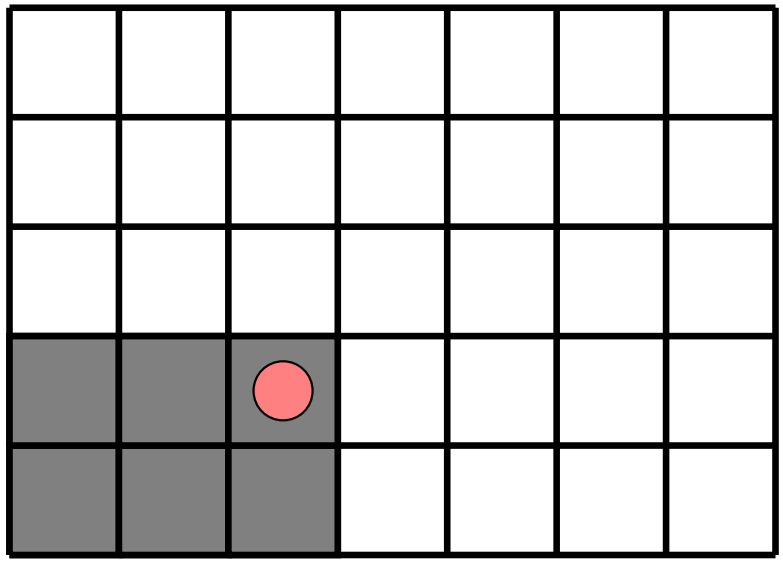
Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



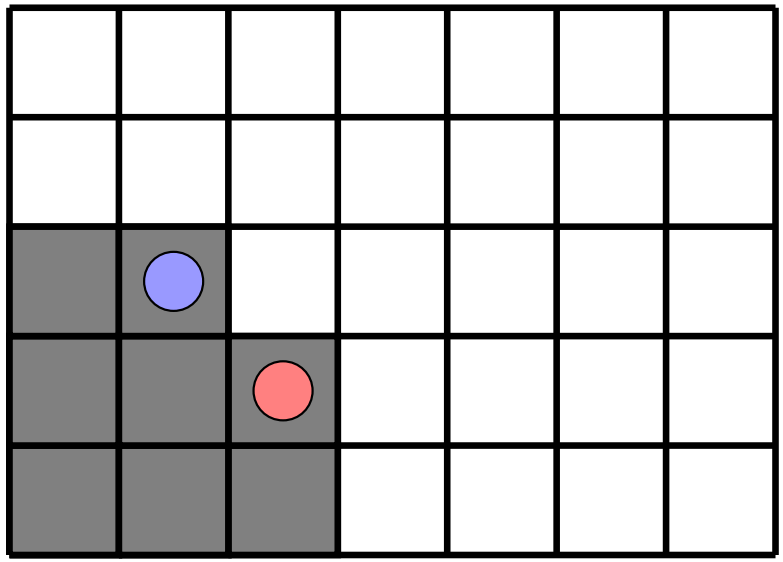
Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



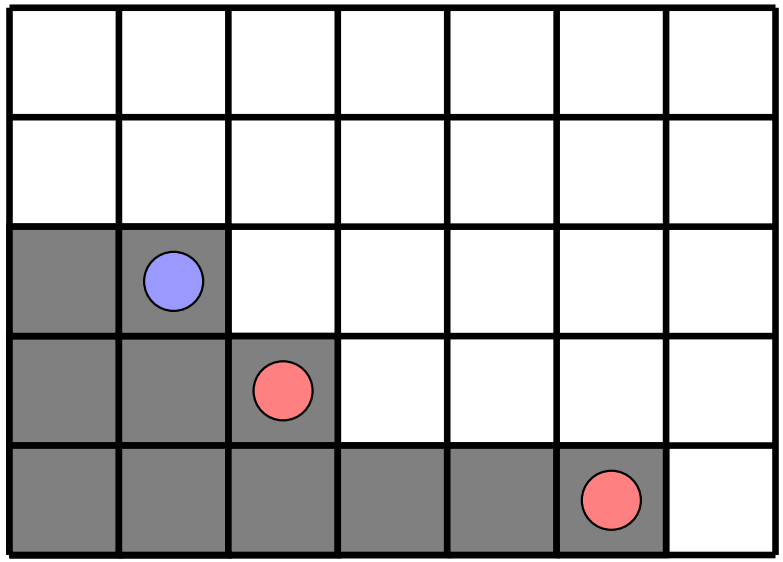
Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



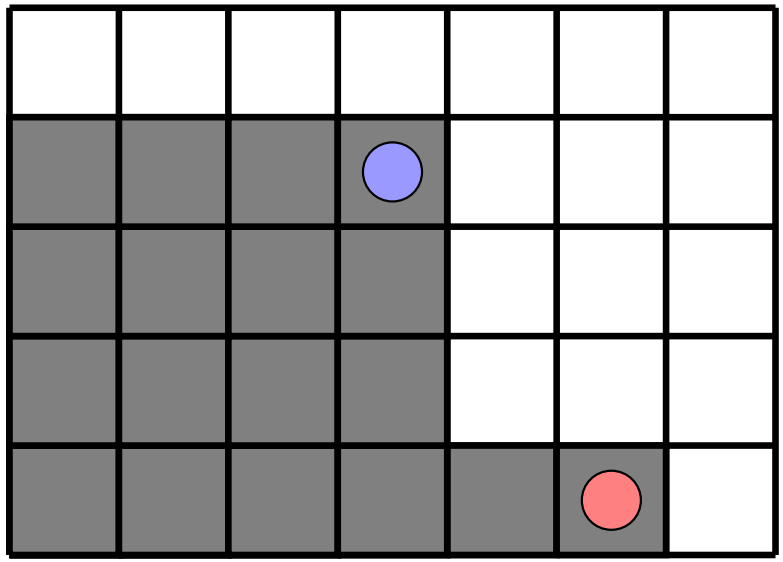
Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



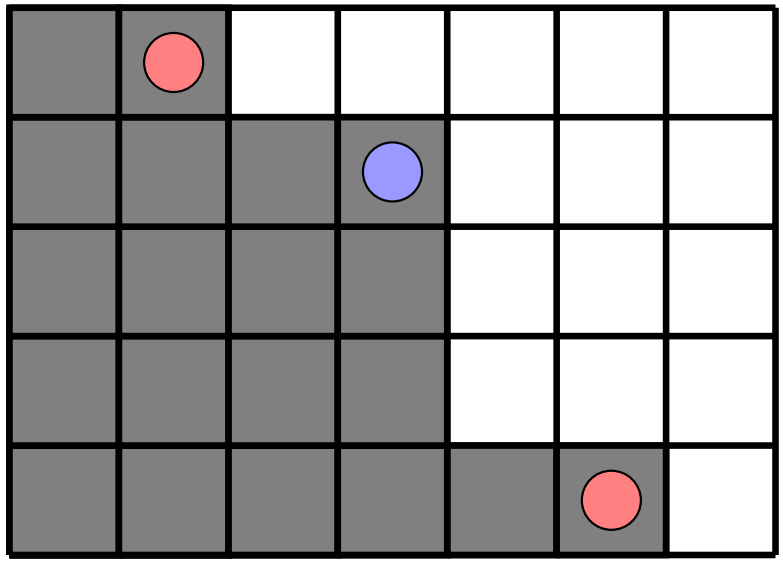
Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



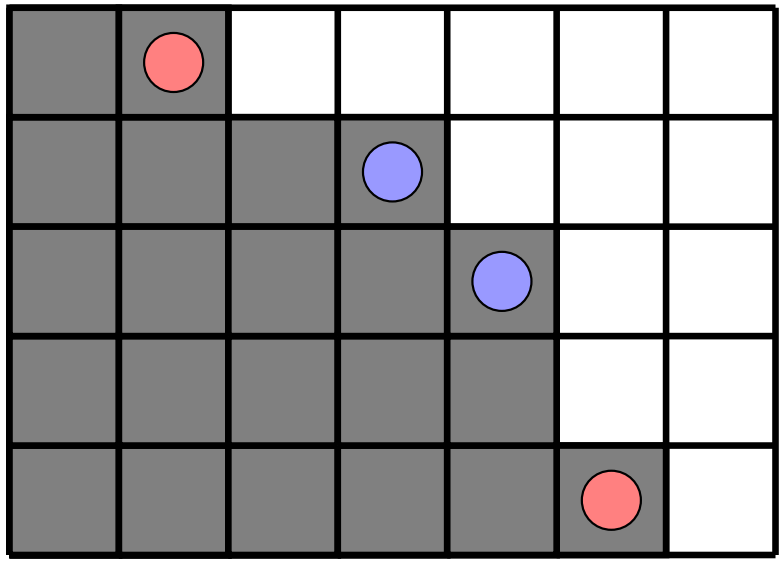
Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



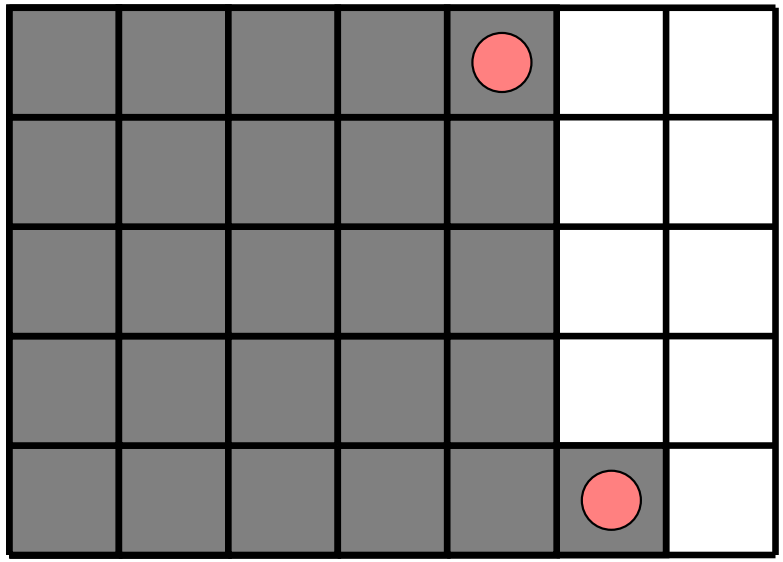
Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



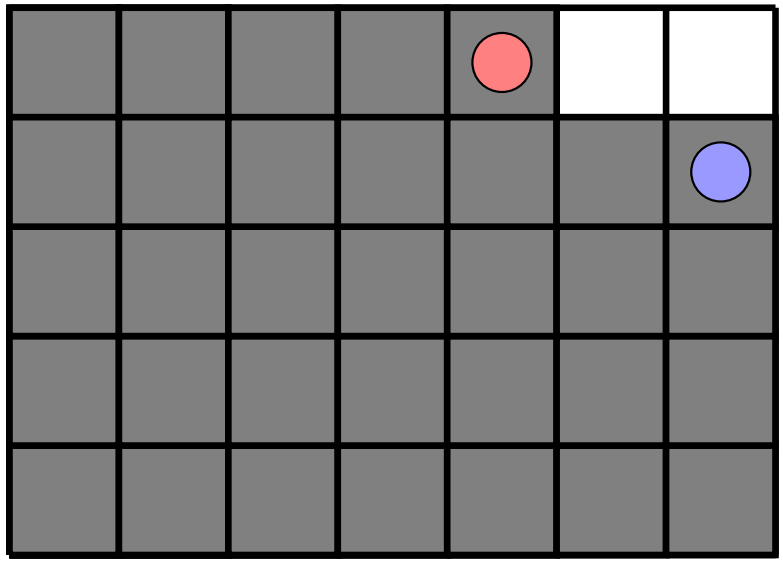
Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



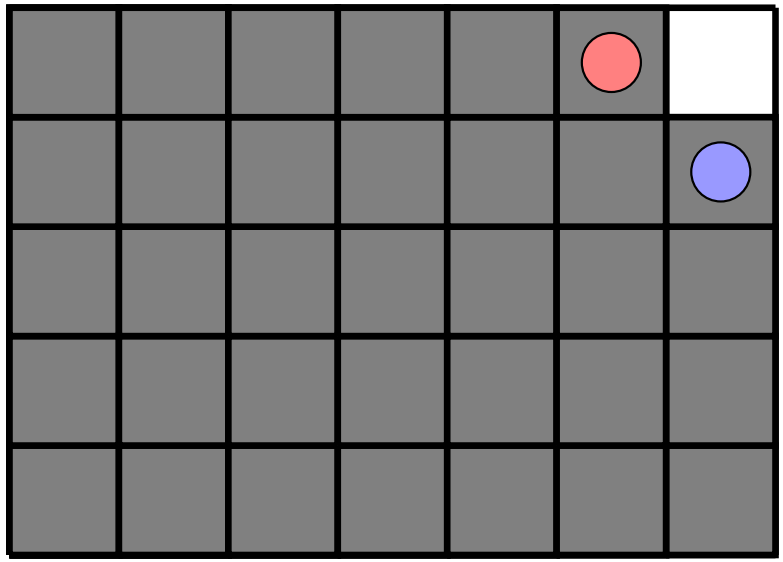
Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



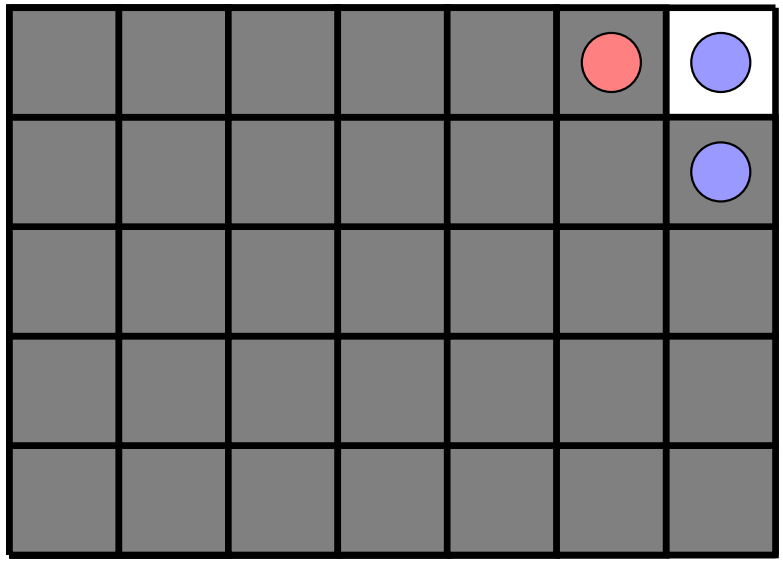
Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



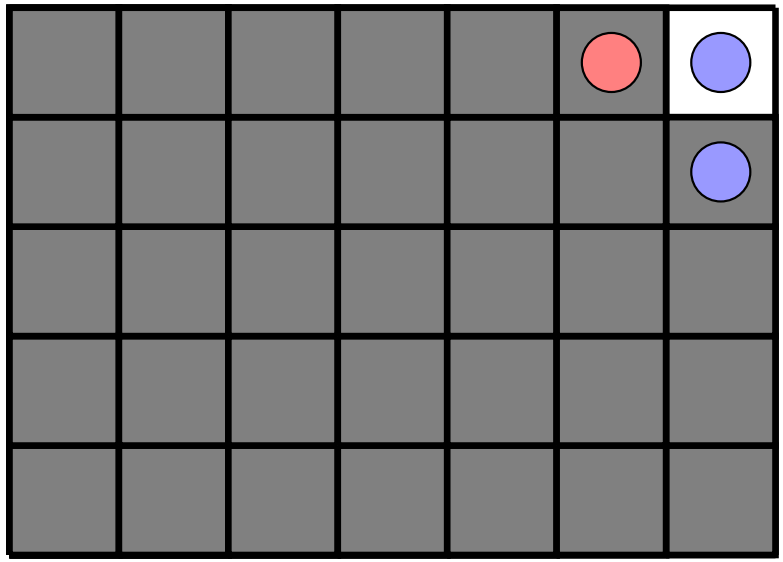
Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



Just for Fun...

Players play tokens in turn, pushing the frontier towards the upper right corner. The player who has to place the token on the upper right field loses:



Challenge: Show that one of the players (the one who starts or the other) has a winning strategy for every size of the board.

The proof is very short and uses positional determinacy of reachability games. But it does not provide the strategy...

1 Basics

2 Positional Strategies: Reachability and Attractors

3 Büchi Games

4 Muller Games and Finite Memory Strategies

5 Parity Games

- Positional Determinacy
- Algorithms for Parity Games
- Reducing Muller Games to Parity Games

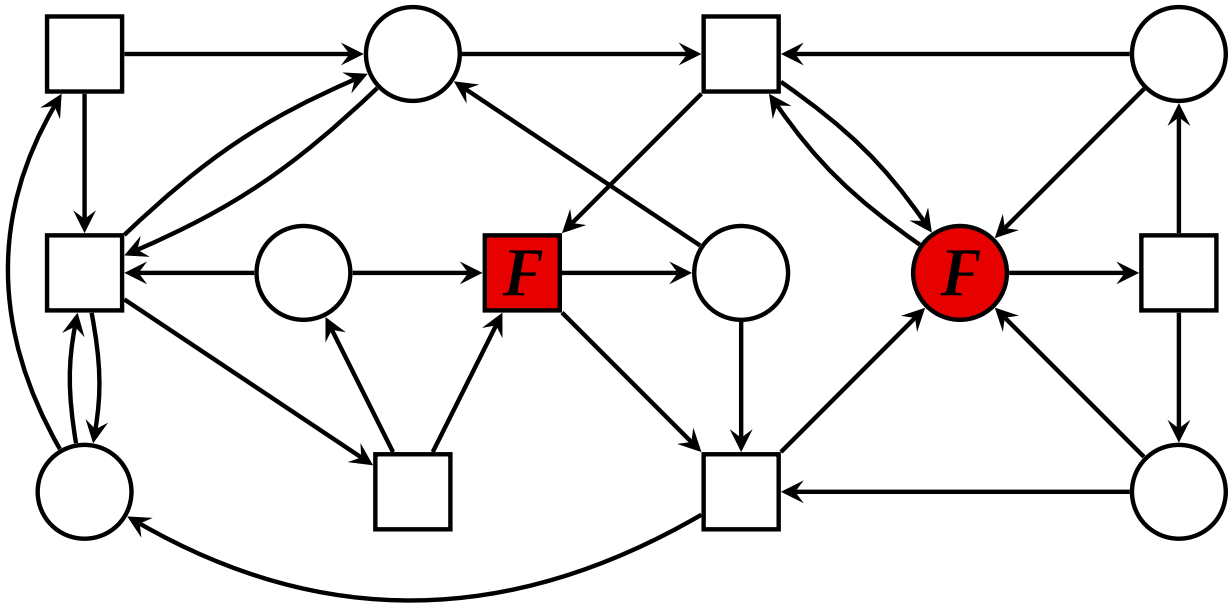
6 Applications

Büchi condition

- A winning condition $Win \subseteq C^\omega$ is called a Büchi condition if there is $D \subseteq C$ such that $\alpha \in Win$ iff $\mathrm{Inf}(c(\alpha)) \cap D \neq \emptyset$, i.e., iff some color from D occurs infinitely often.
- The objective for Eva in a Büchi game is to reach infinitely often some vertex with color D.
- Normally, we specify a Büchi game as $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to see infinitely often.
- Origin of the name: J. Richard Büchi. *On a decision method in restricted second order arithmetic.* 1962

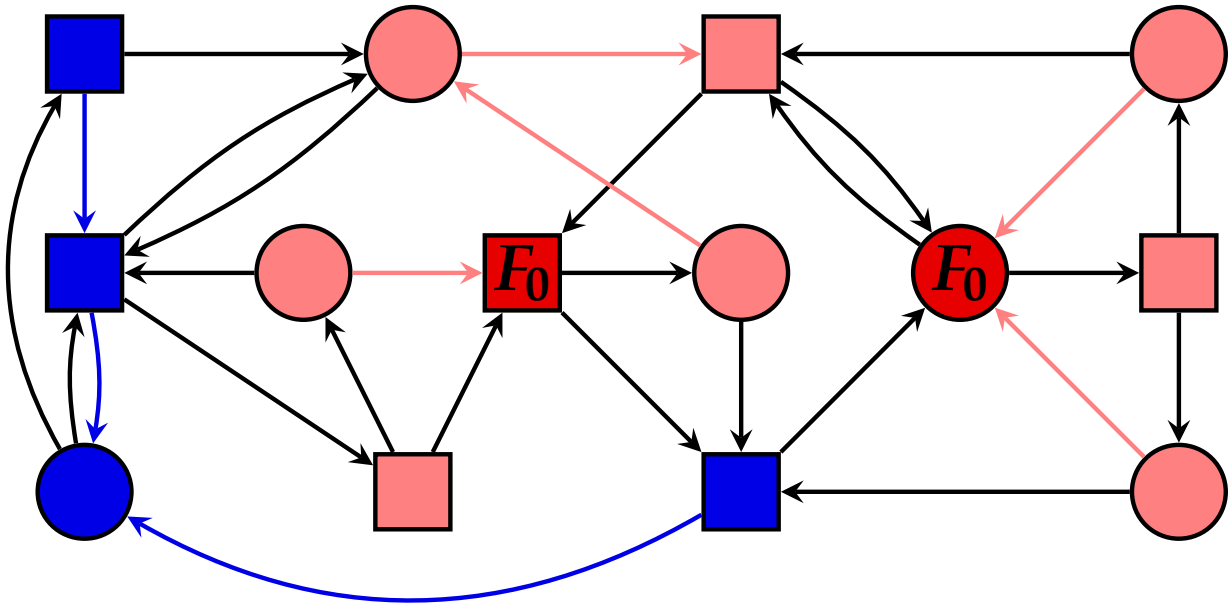
Solving Büchi Games – Idea

Iterating the attractor:



Solving Büchi Games – Idea

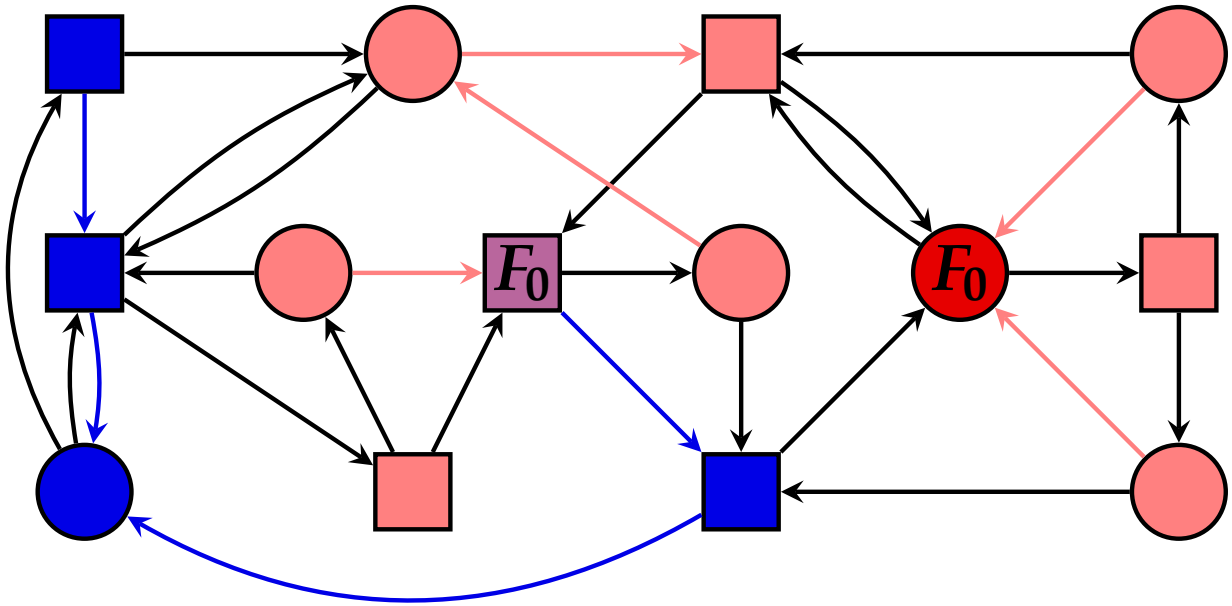
Iterating the attractor:



- $F_0 = F$
- $W^0_\forall =$ **Adam can avoid** F_0

Solving Büchi Games – Idea

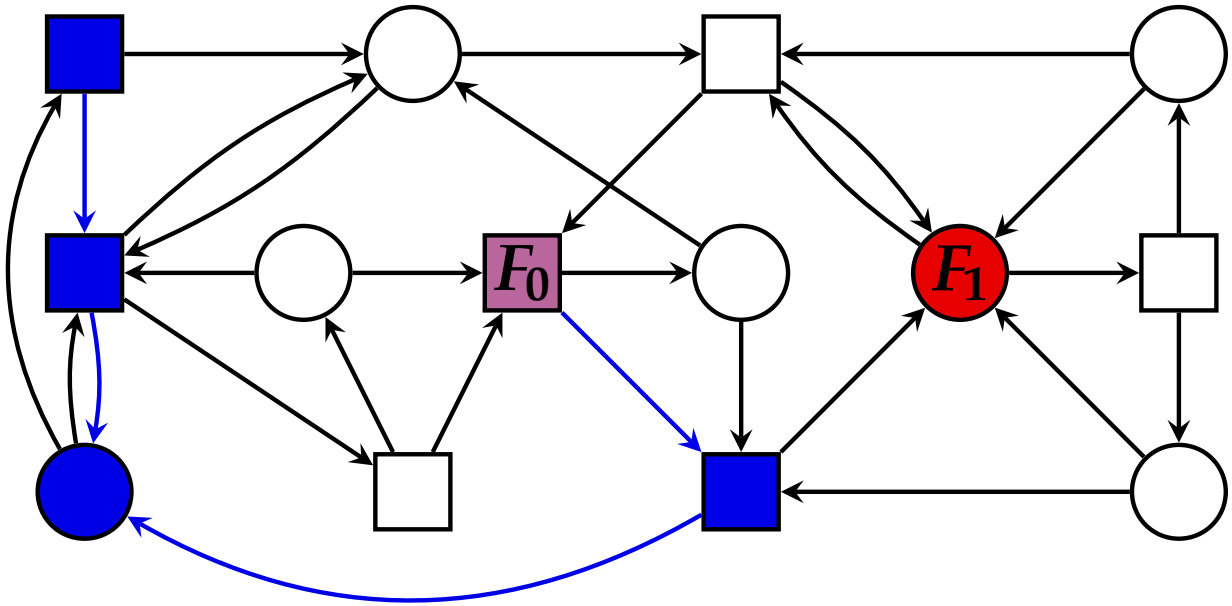
Iterating the attractor:



- $F_0 = F$
- $W_\forall^0 =$ **Adam can avoid** F_0
- $F_1 = F_0 \setminus Pre_\forall(W_\forall^0)$

Solving Büchi Games – Idea

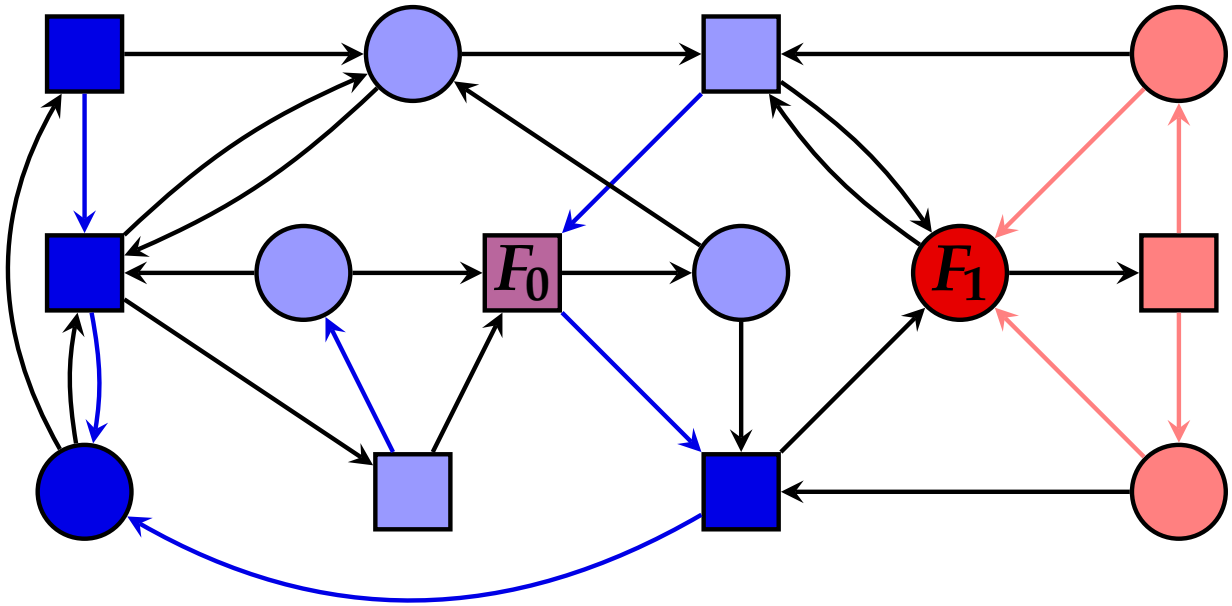
Iterating the attractor:



- $F_0 = F$
- $W_\forall^0 =$ **Adam can avoid** F_0
- $F_1 = F_0 \setminus Pre_\forall(W_\forall^0)$

Solving Büchi Games – Idea

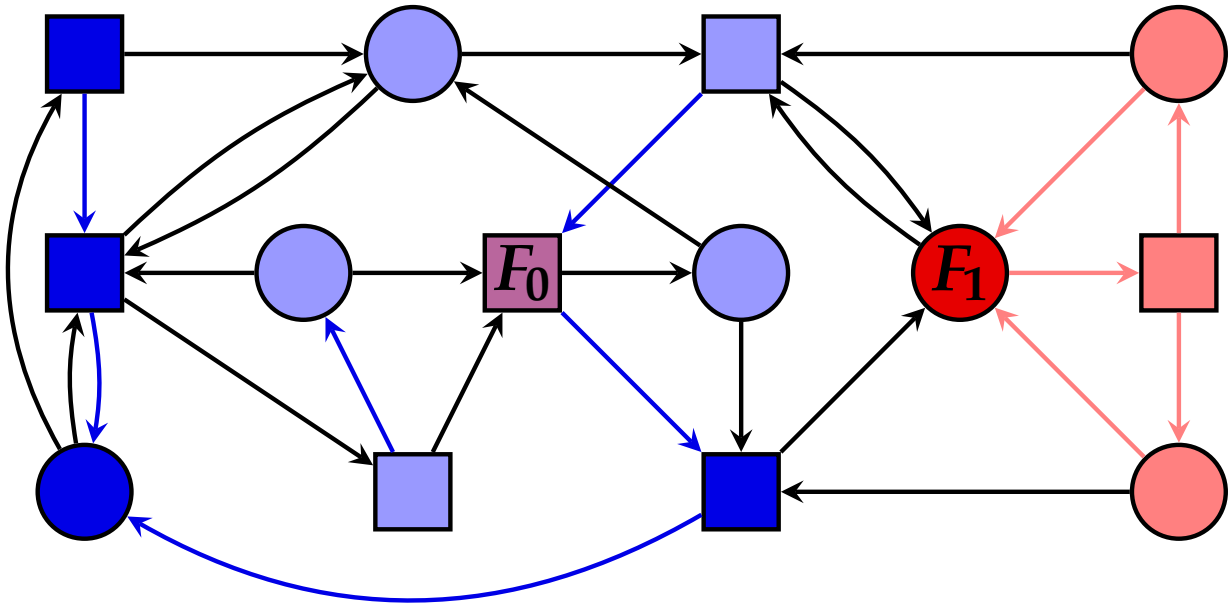
Iterating the attractor:



- $F_0 = F$
- $W_\forall^0 =$ **Adam can avoid** F_0
- $F_1 = F_0 \setminus Pre_\forall(W_\forall^0)$
- $W_\forall^1 =$ **Adam can avoid** F_1

Solving Büchi Games – Idea

Iterating the attractor:



- $F_0 = F$
- $W_\forall^0 =$ **Adam can avoid** F_0
- $F_1 = F_0 \setminus Pre_\forall(W_\forall^0)$
- $W_\forall^1 =$ **Adam can avoid** F_1
- $F_2 = F_1 \setminus Pre_\forall(W_\forall^1) = F_1$

Solving Büchi Games – Finite Graphs

Start with $F_0 := F$. In Iteration i:

- **Let $\mathcal{G}_i = (G, F_i)$ be the reachability game for F_i.**
- **Let $W_\forall^i$ be the winning area of Adam in $\mathcal{G}_i$.**
- **Let $F_{i+1} = F_i \setminus Pre_\forall(W_\forall^i)$.**

Solving Büchi Games – Finite Graphs

Start with $F_0 := F$. In Iteration i:

- **Let $\mathcal{G}_i = (G, F_i)$ be the reachability game for F_i.**
- **Let $W_\forall^i$ be the winning area of Adam in $\mathcal{G}_i$.**
- **Let $F_{i+1} = F_i \setminus Pre_\forall(W_\forall^i)$.**

$$F_0 \supseteq F_1 \supseteq F_2 \supseteq \cdots$$

Termination for $F_{i+1} = F_i$.

Illustration of Winning Strategies

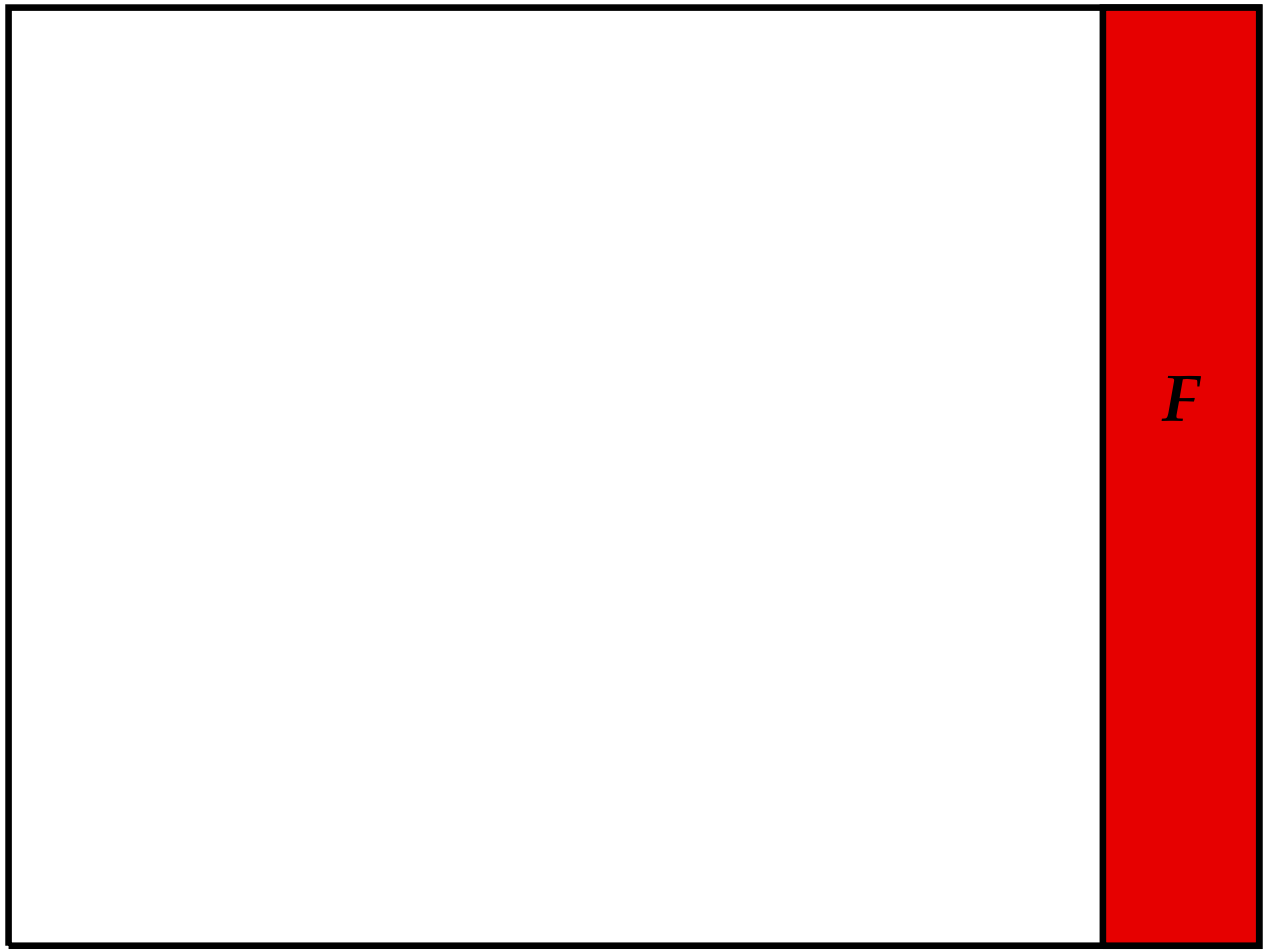


Illustration of Winning Strategies

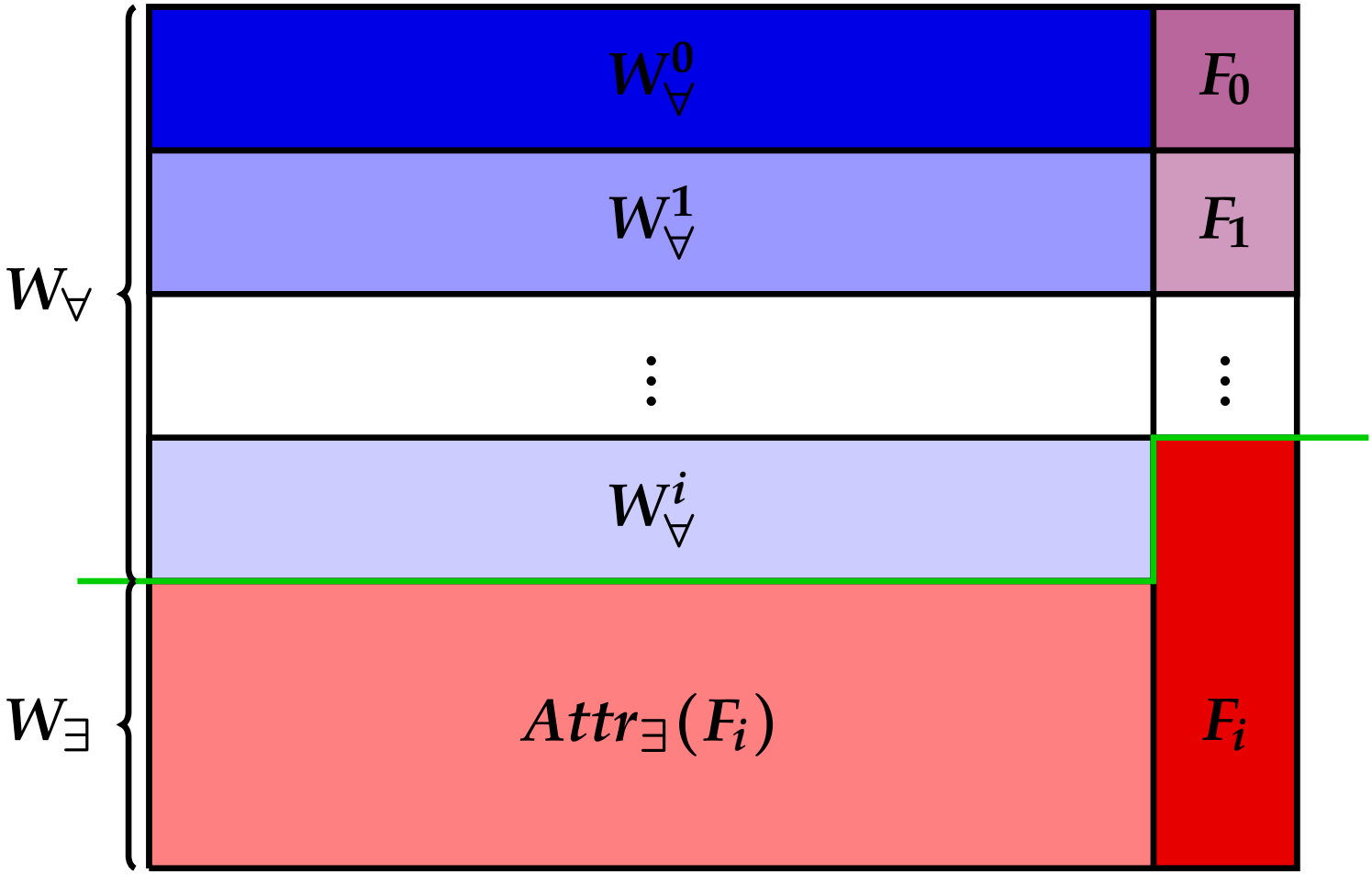


Illustration of Winning Strategies

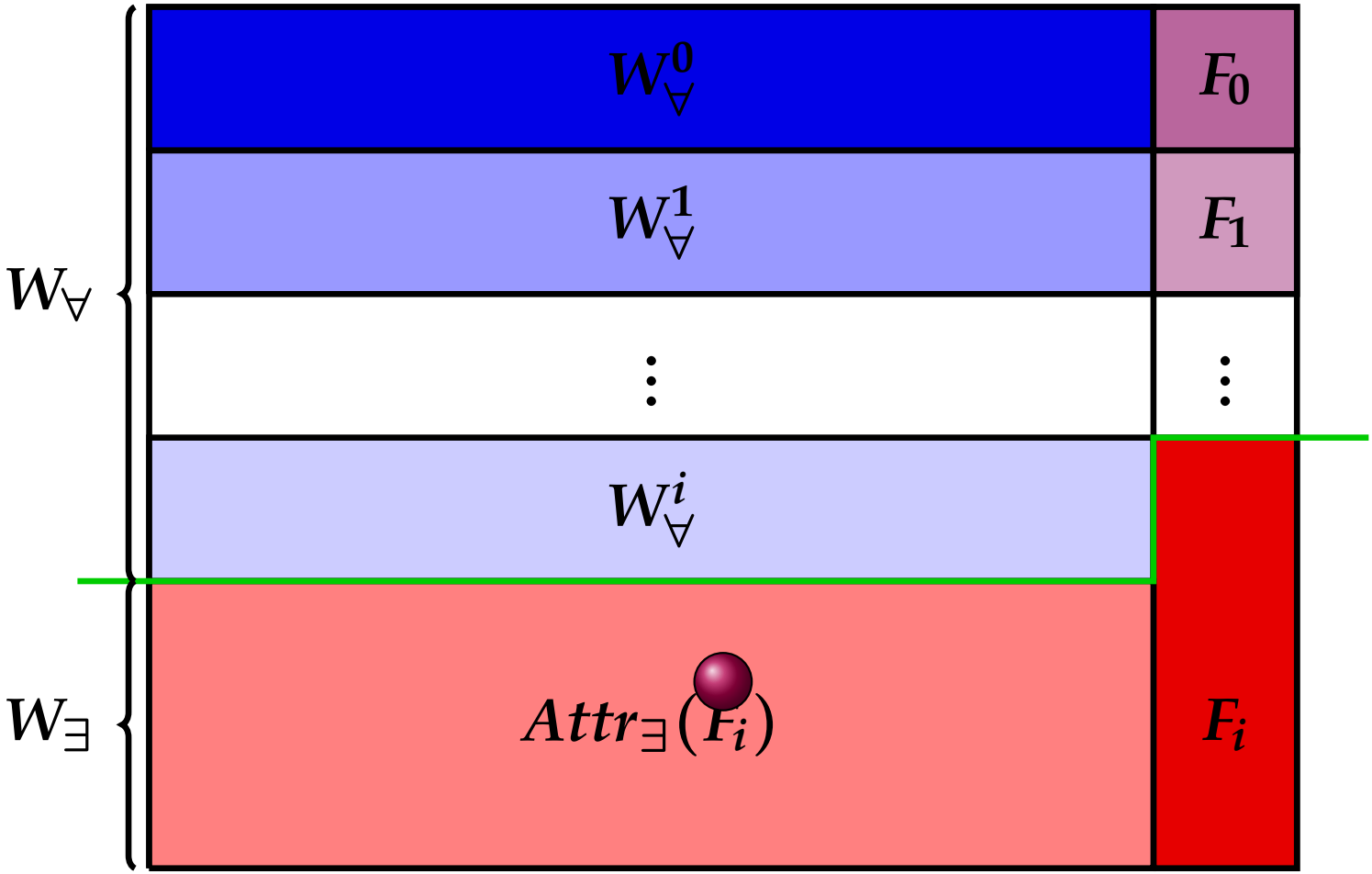


Illustration of Winning Strategies

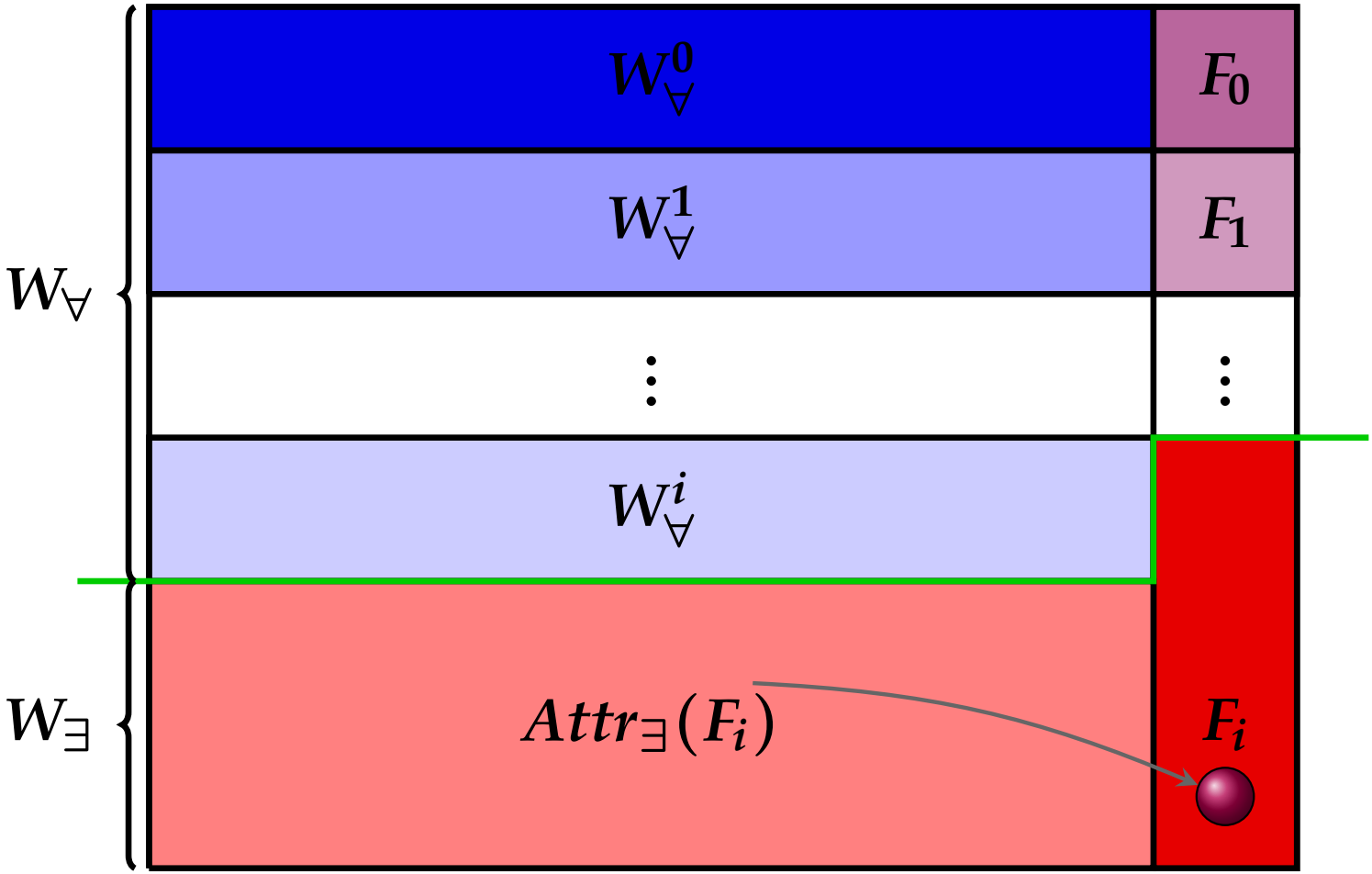


Illustration of Winning Strategies

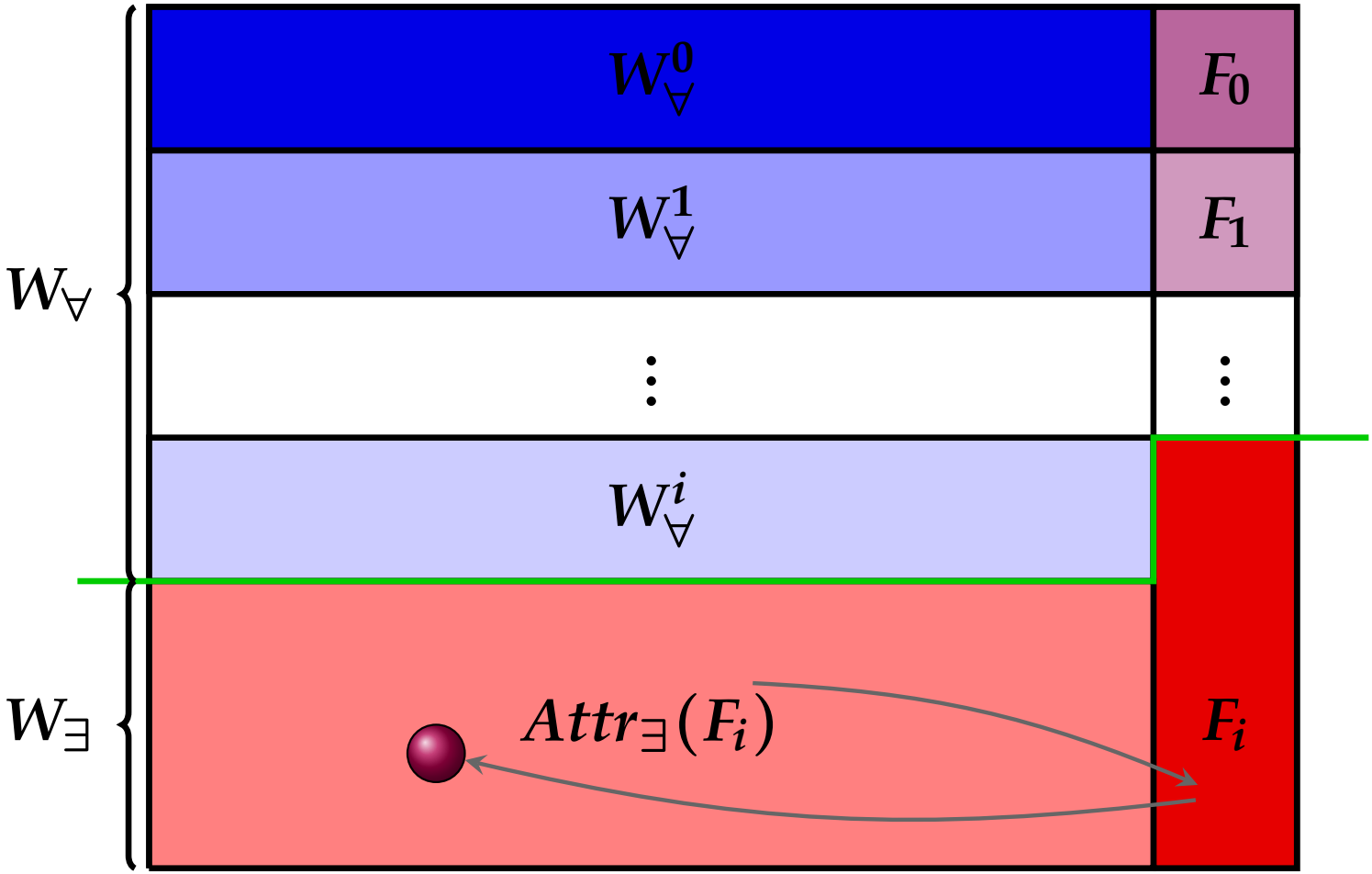


Illustration of Winning Strategies

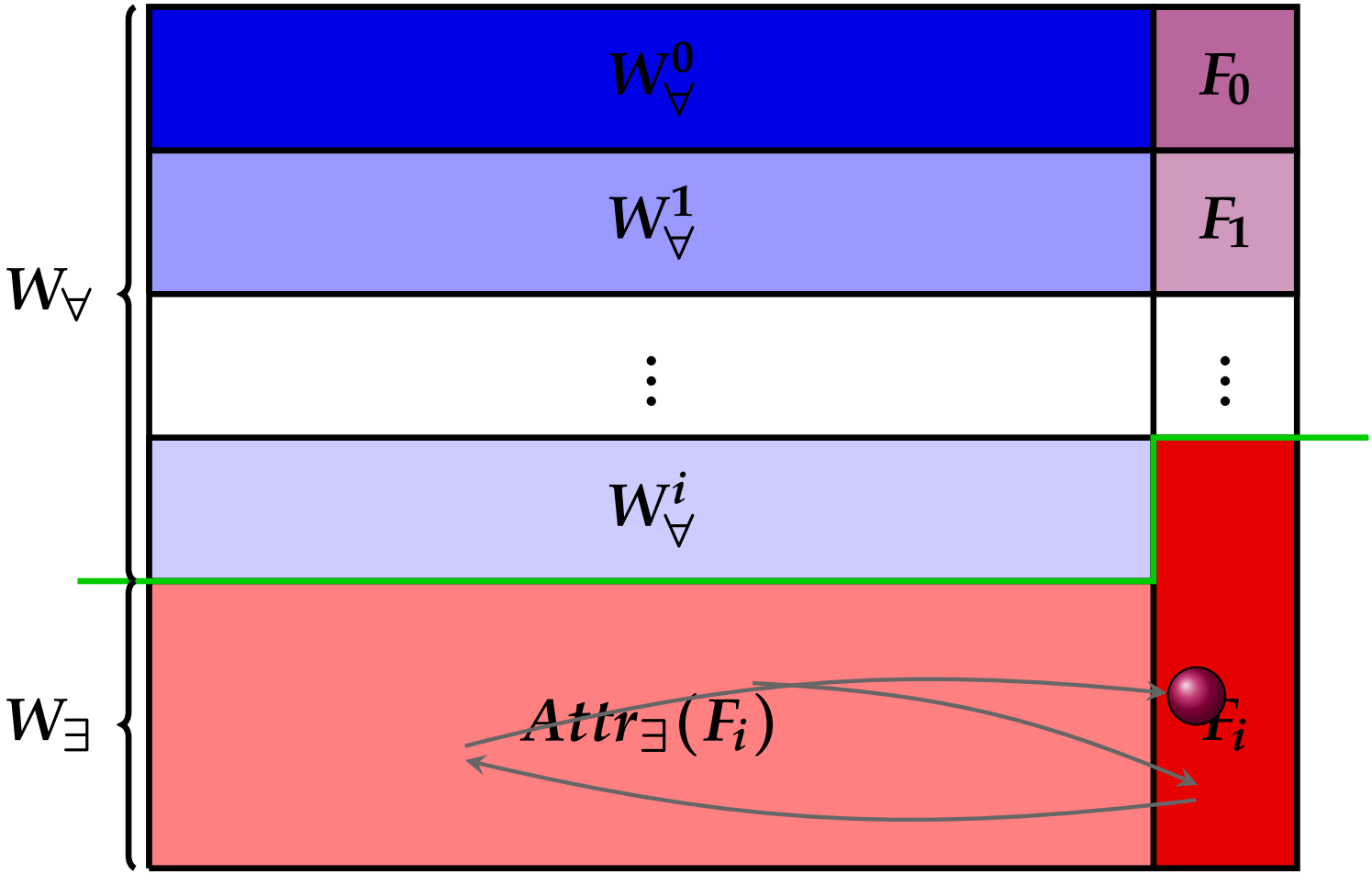


Illustration of Winning Strategies

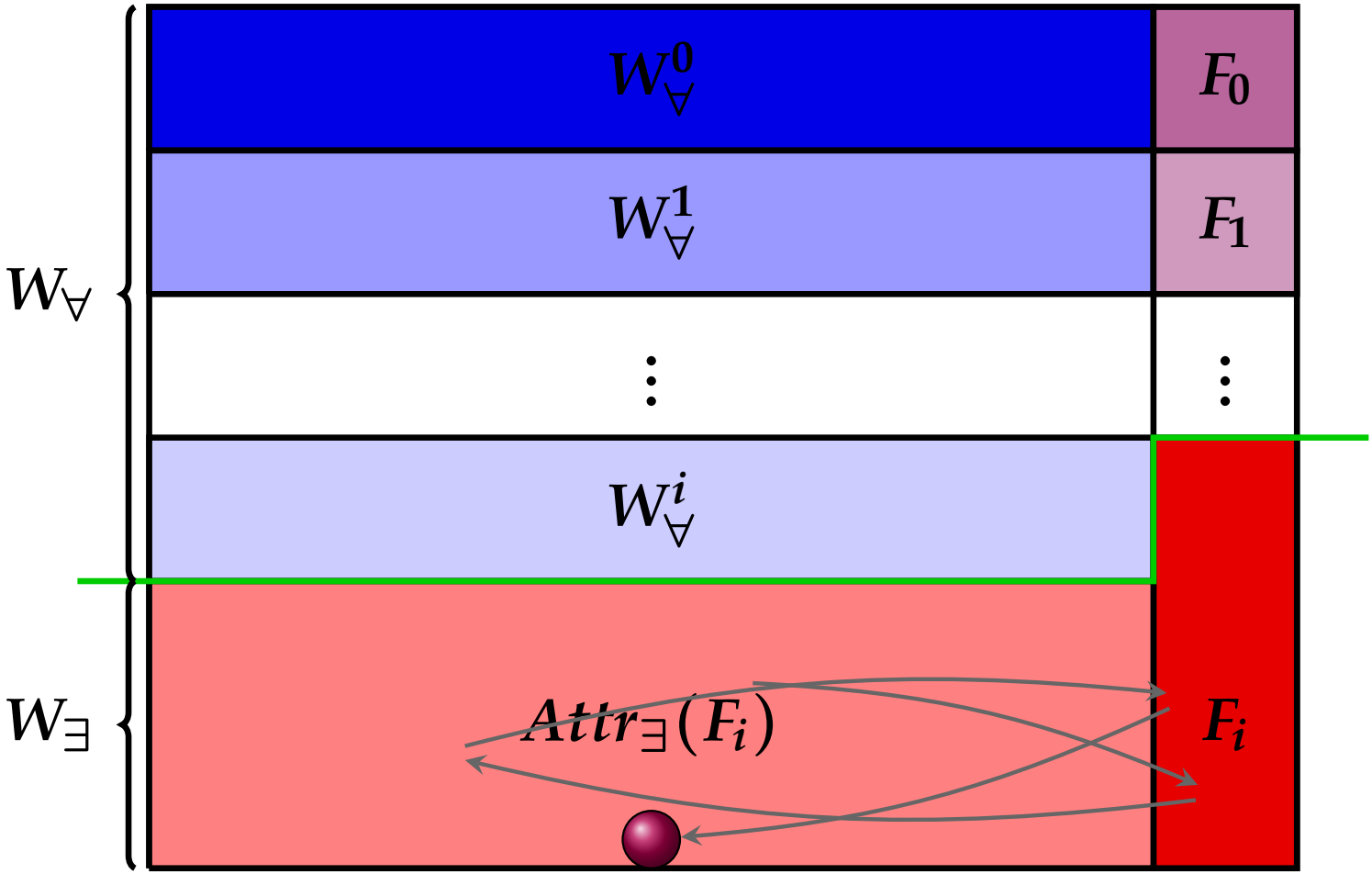


Illustration of Winning Strategies

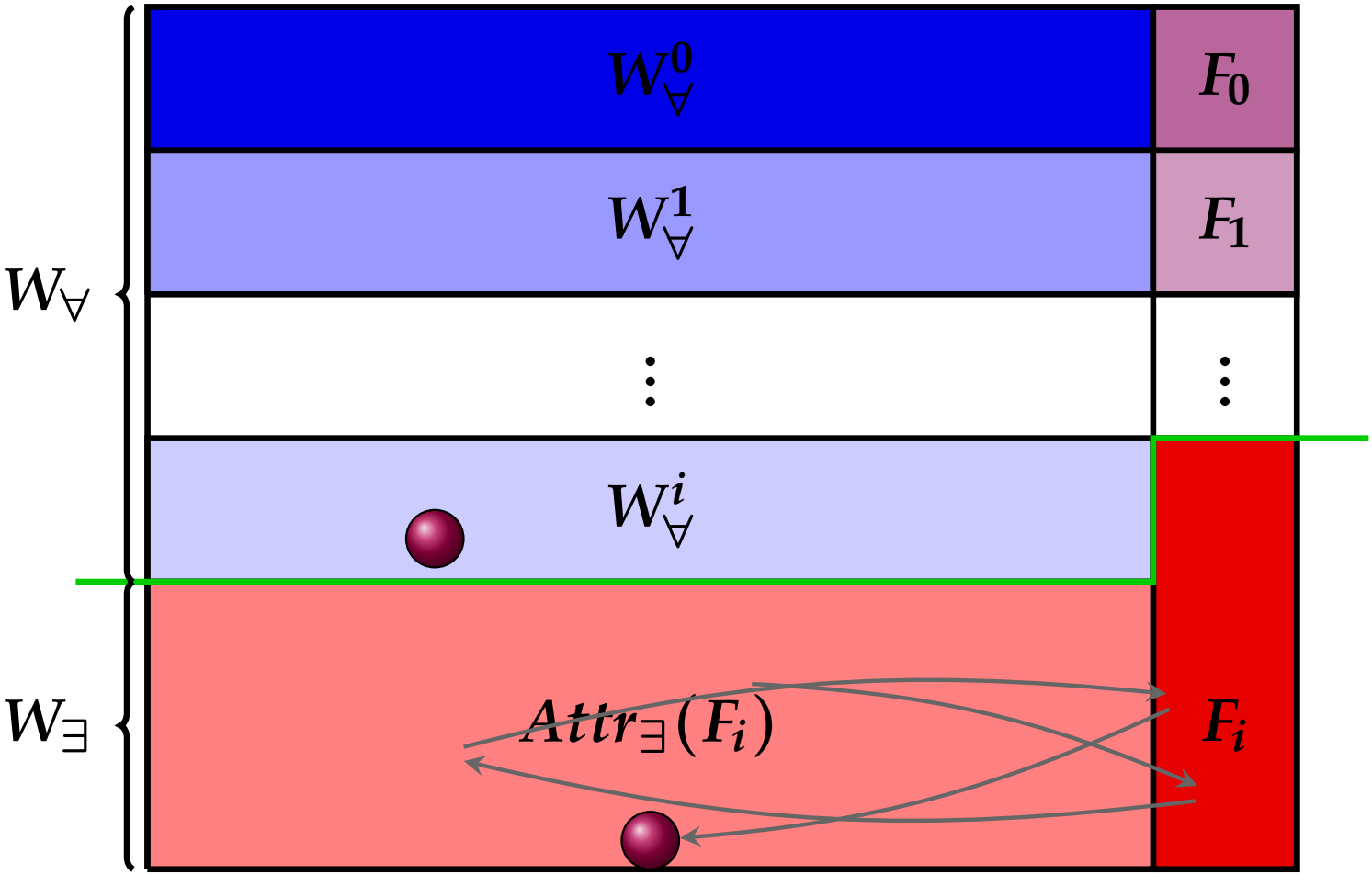


Illustration of Winning Strategies

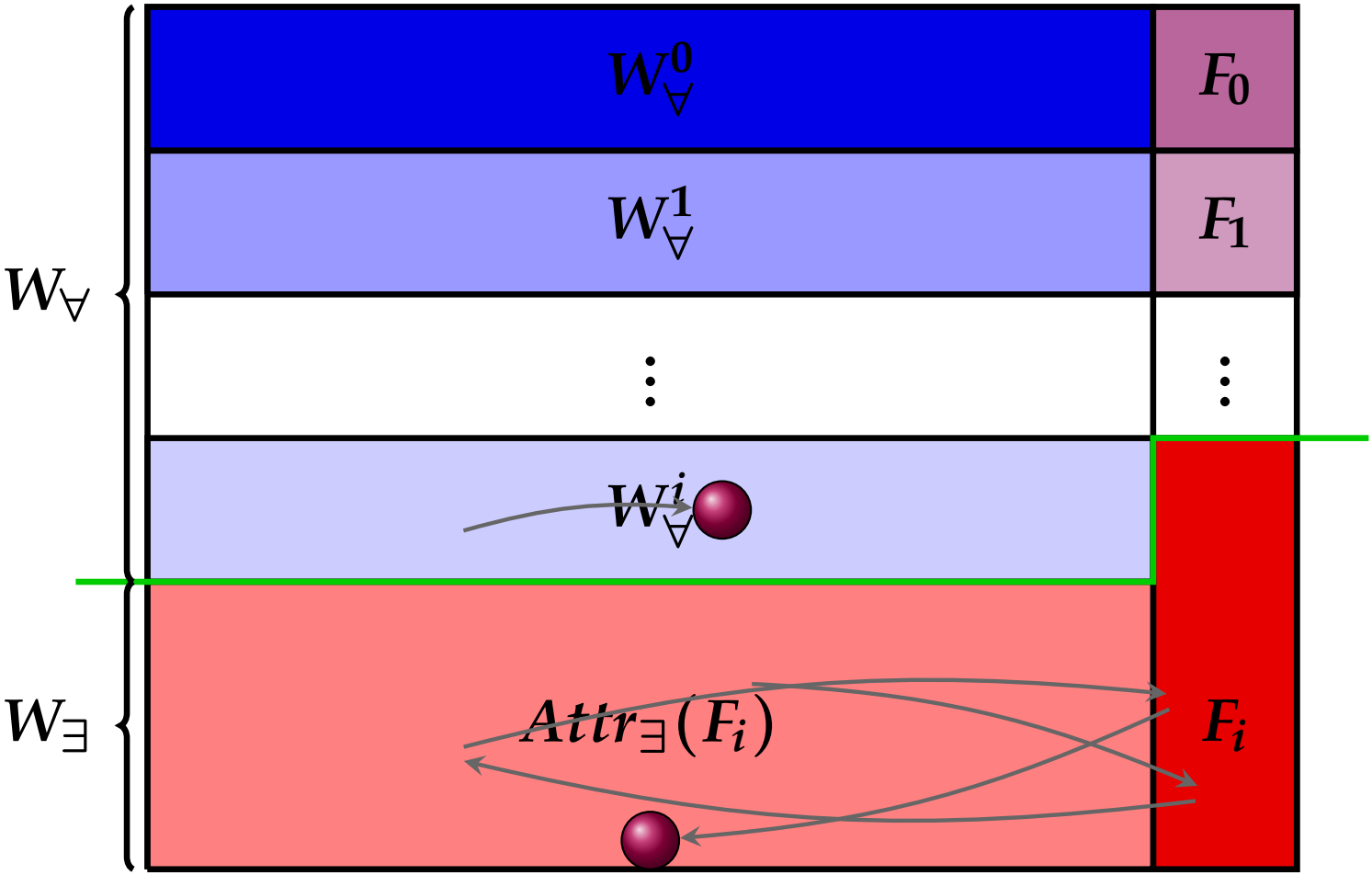


Illustration of Winning Strategies

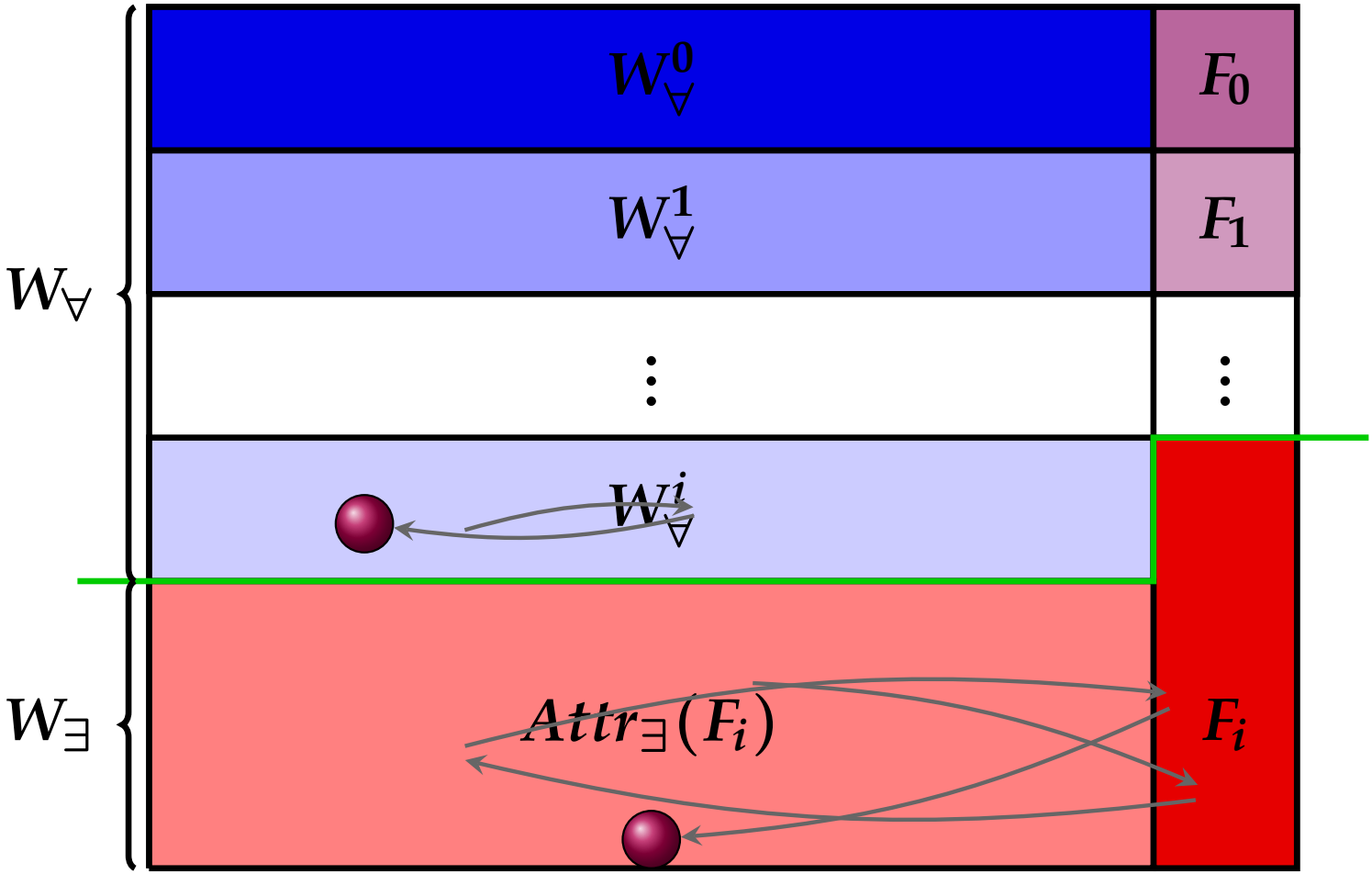


Illustration of Winning Strategies

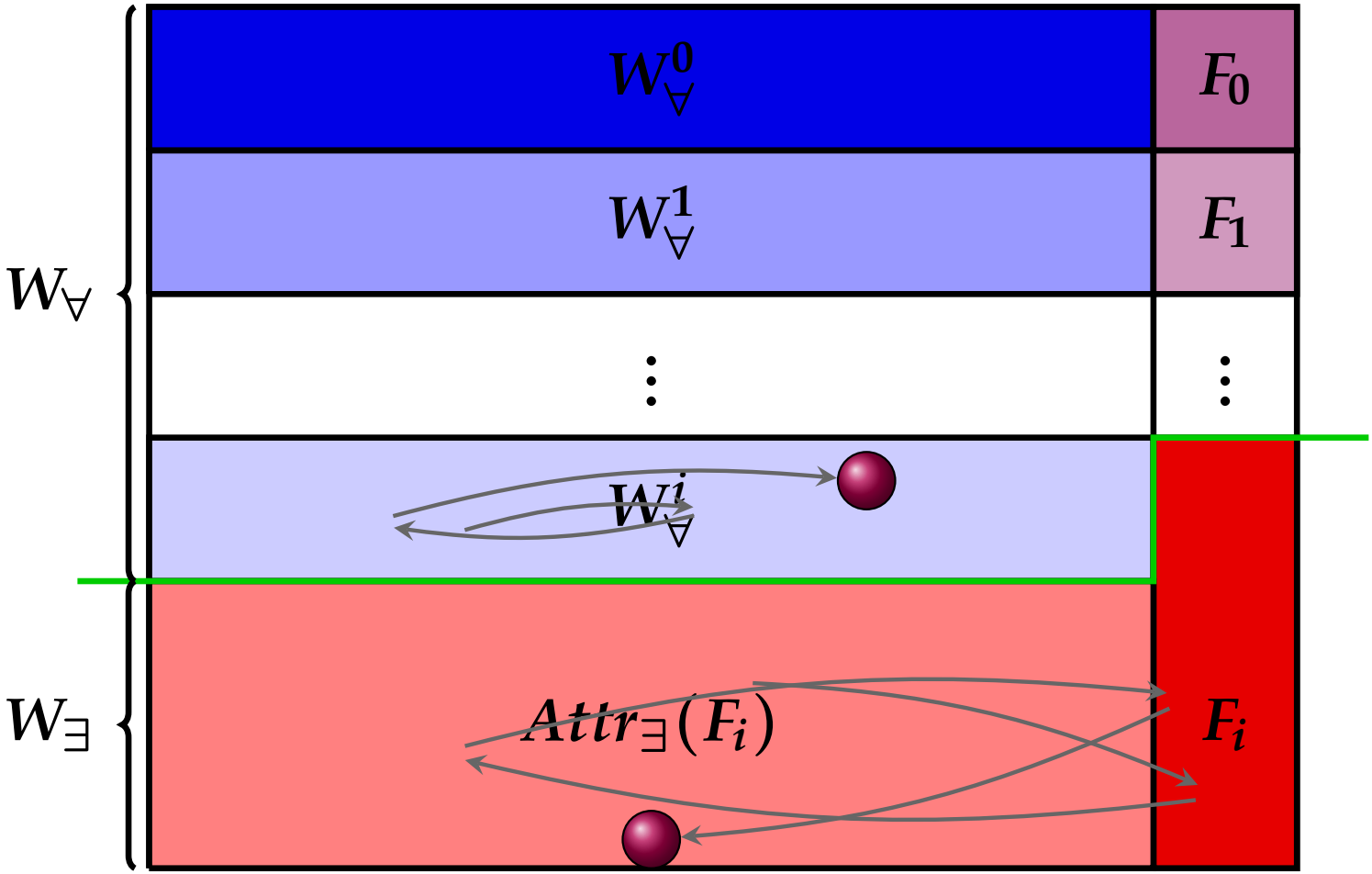


Illustration of Winning Strategies

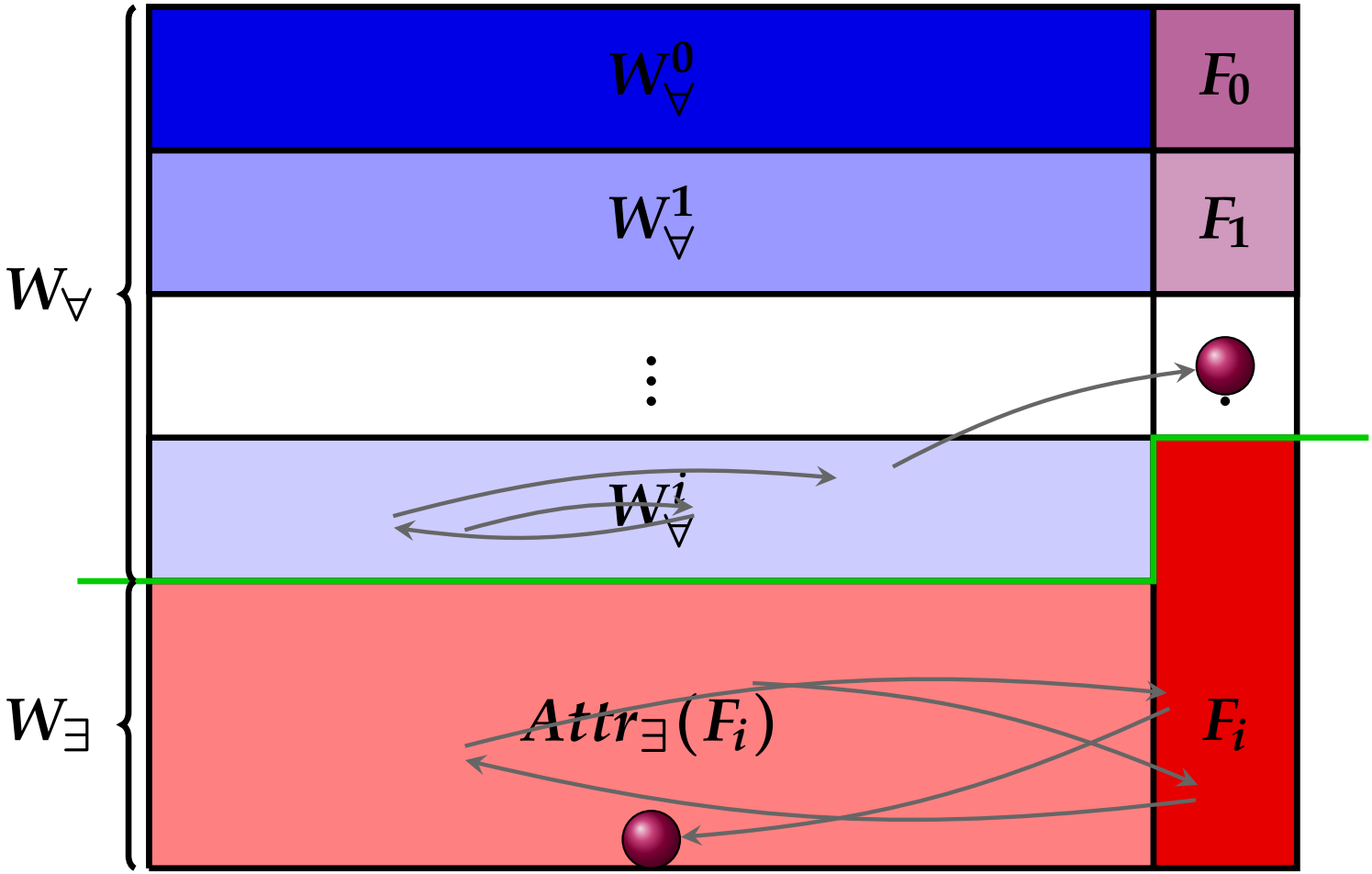


Illustration of Winning Strategies

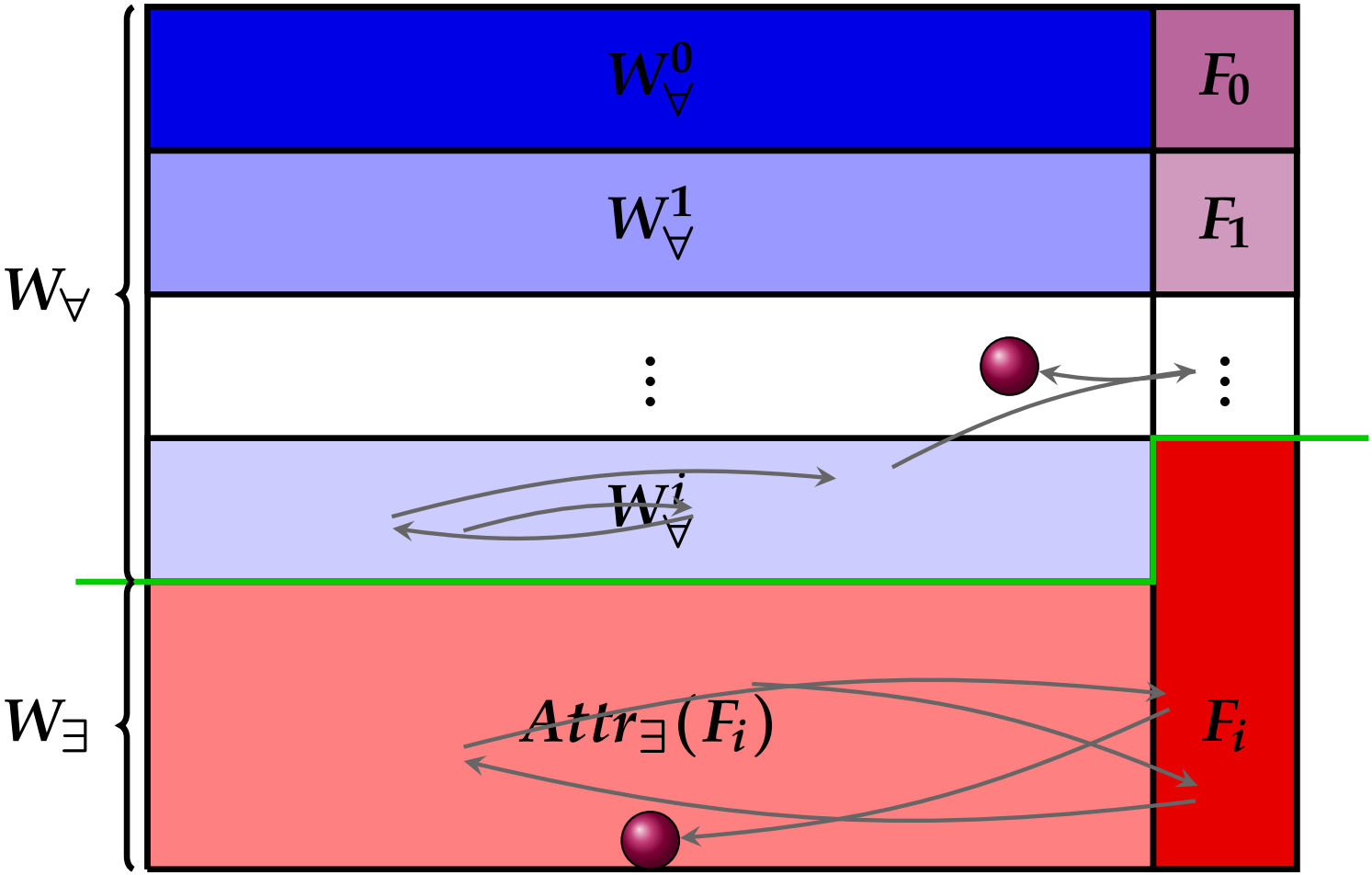


Illustration of Winning Strategies

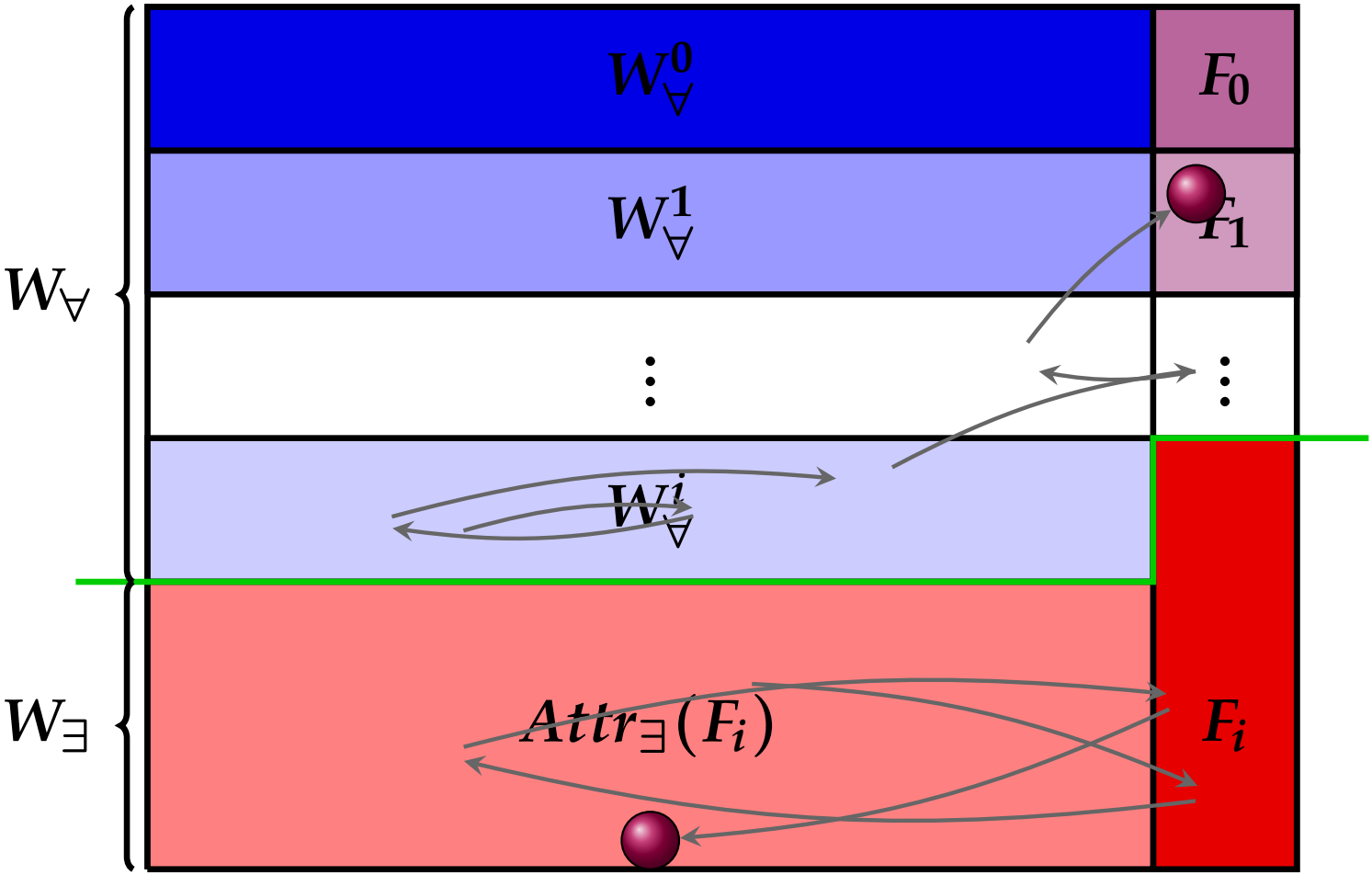


Illustration of Winning Strategies

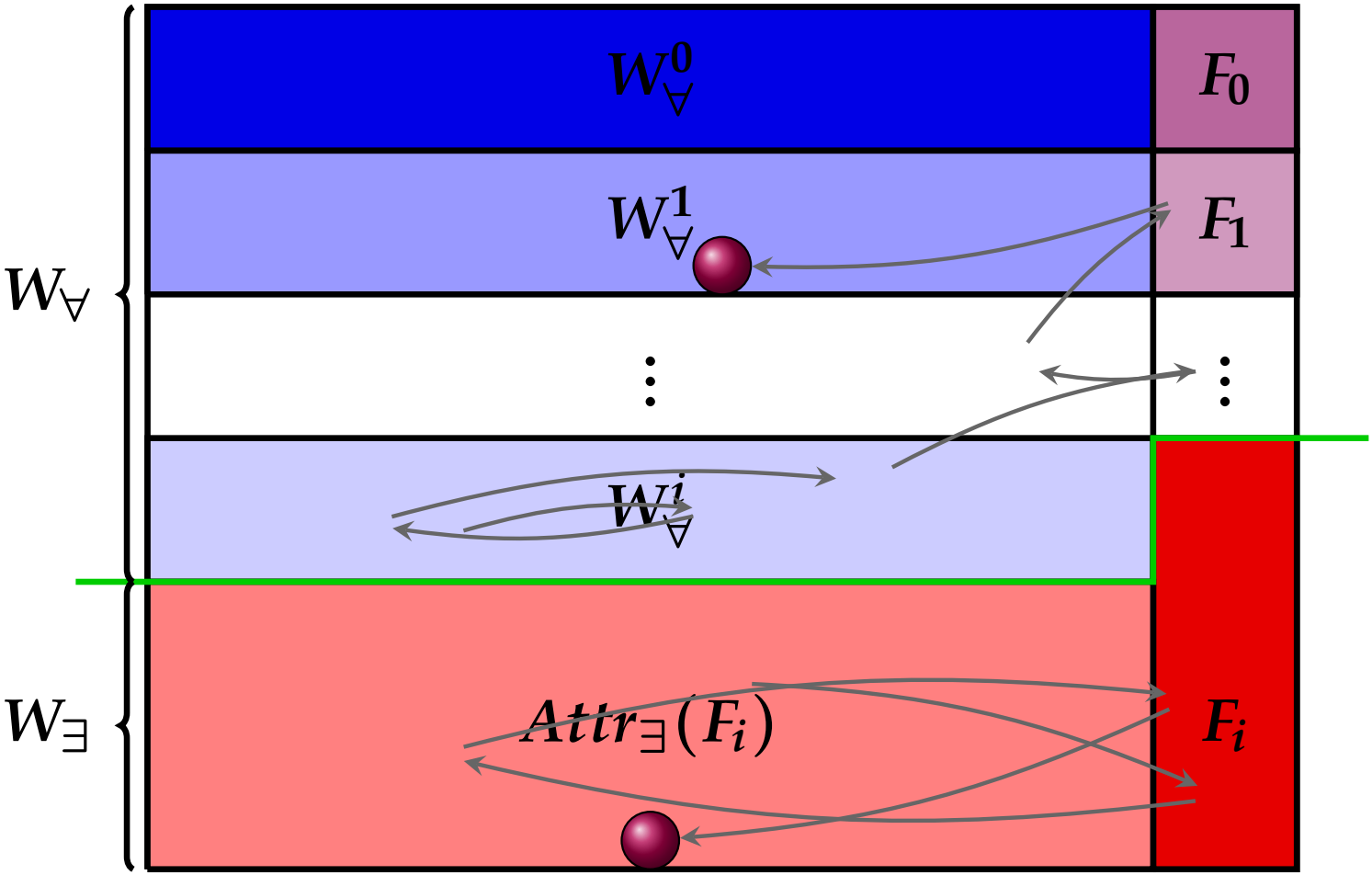


Illustration of Winning Strategies

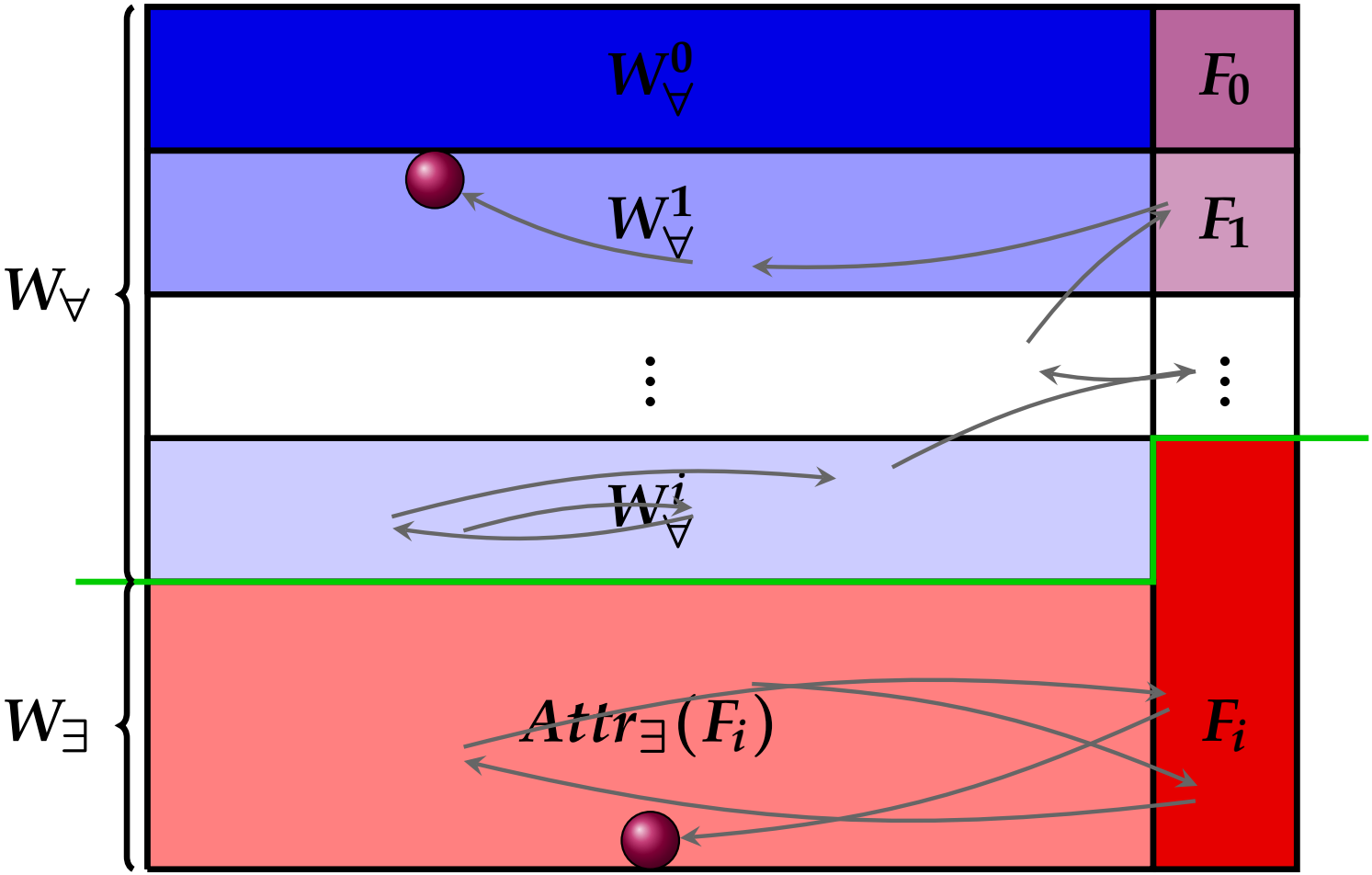


Illustration of Winning Strategies

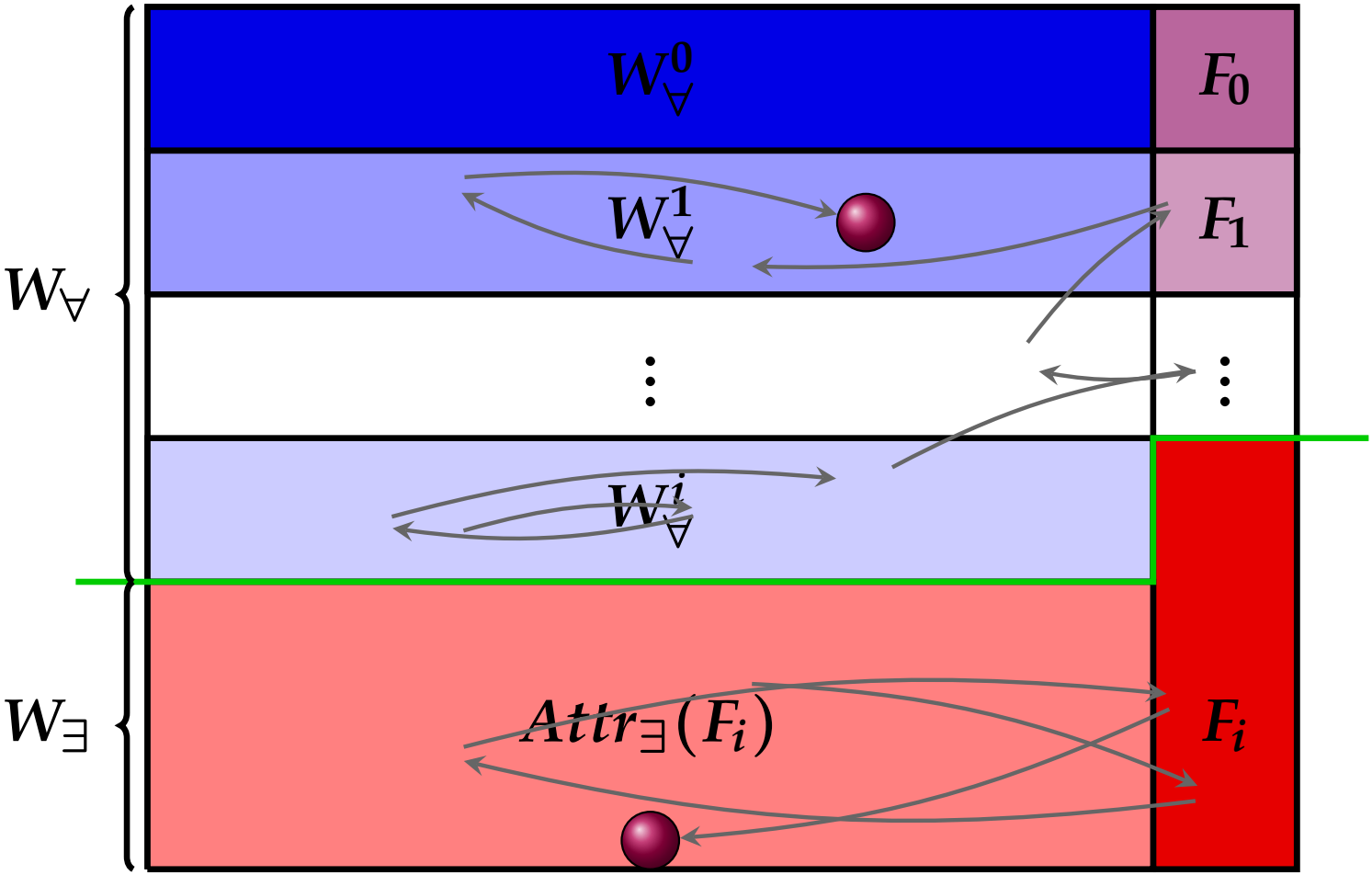


Illustration of Winning Strategies

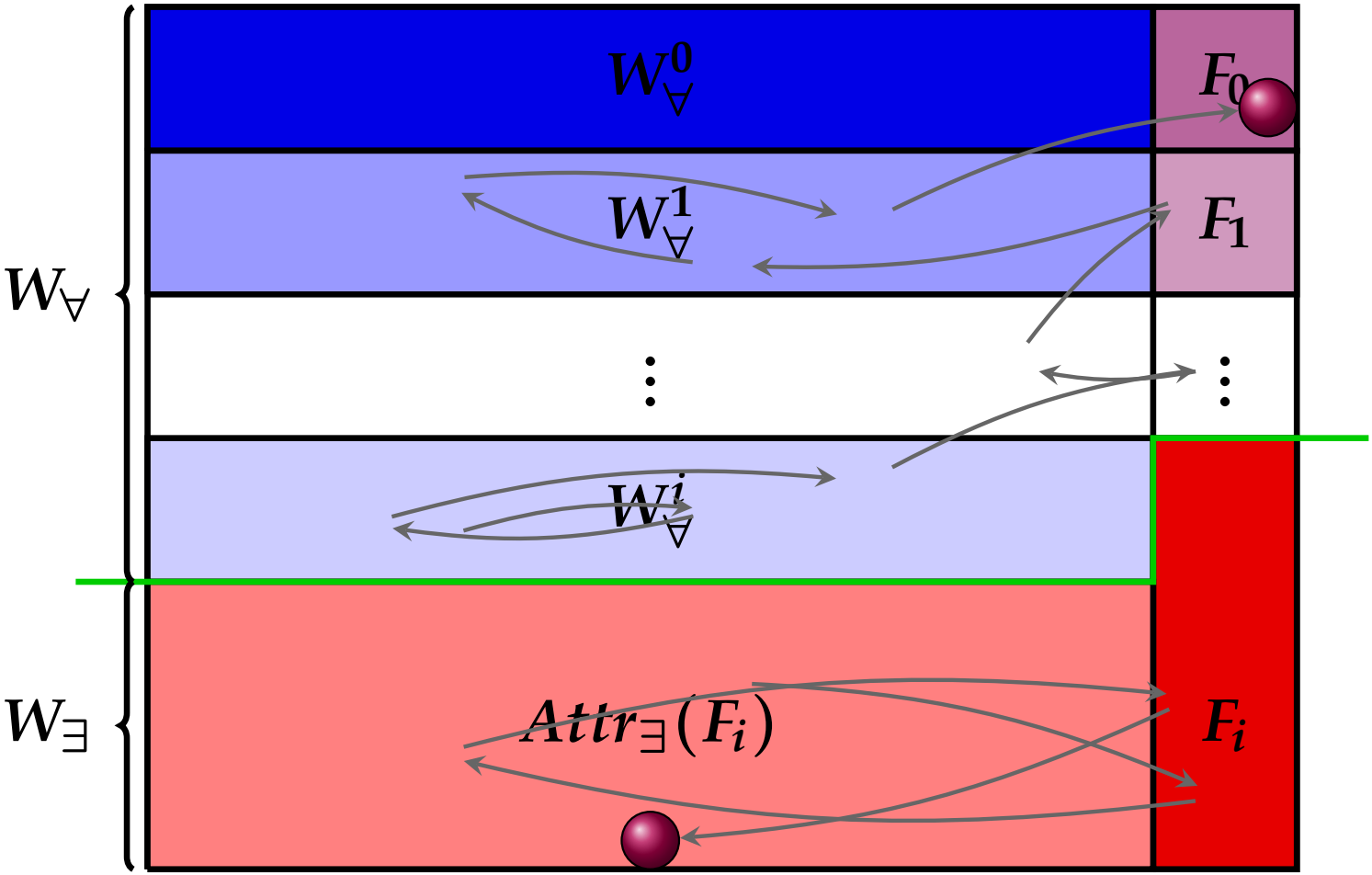
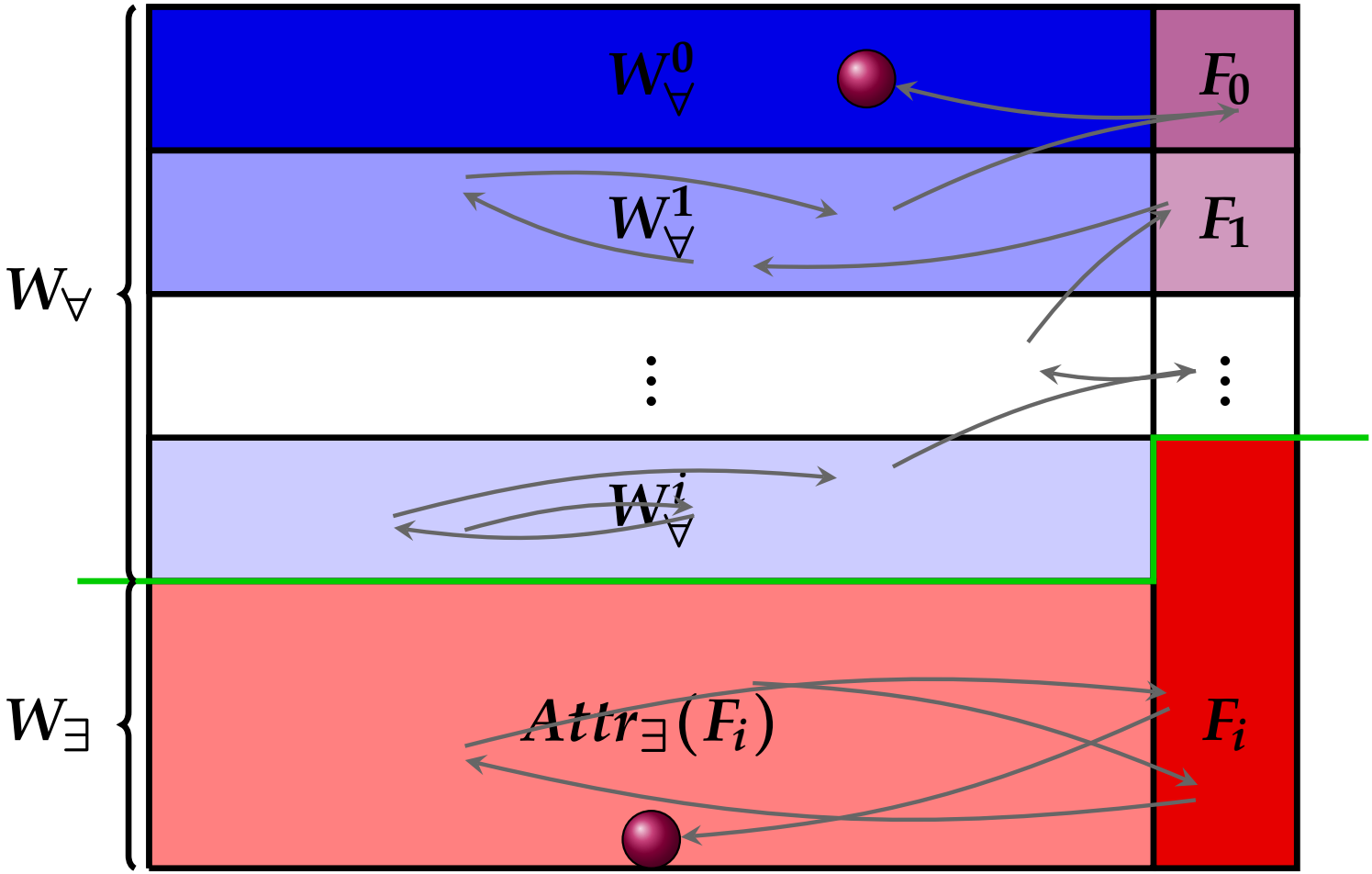
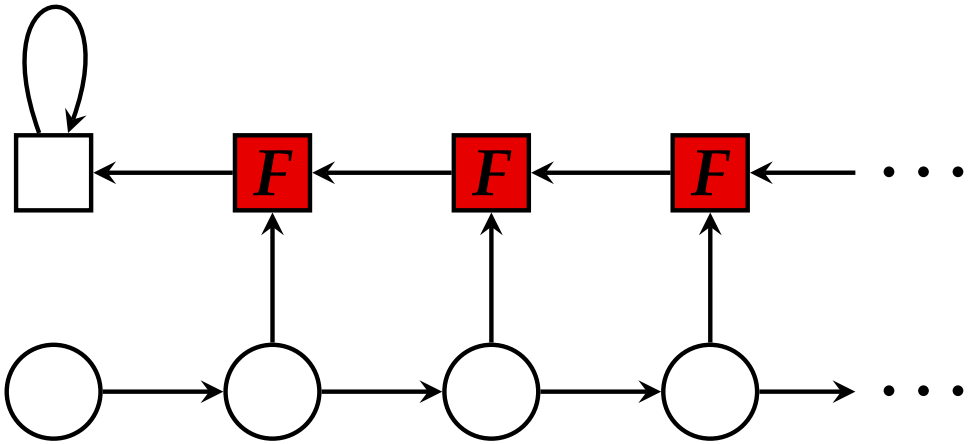


Illustration of Winning Strategies



Remark on Infinite Graphs

For infinite graphs it might be necessary to iterate over ordinals: For a limit ordinal i we set $F_i = \bigcap_{j<i} F_j$.



$$F_\omega = \bigcap_{j<\omega} F_j = \emptyset$$

Determinacy of Büchi Games

Theorem

Büchi games are positionally determined.

Complexity:

- **In each iteration F_i gets smaller**
  $\leadsto$ termination after at most $|F|$ steps
- **Each step requires to solve a reachability game (linear)**
- **Total time complexity is quadratic**

1 Basics

2 Positional Strategies: Reachability and Attractors

3 Büchi Games

4 Muller Games and Finite Memory Strategies

5 Parity Games

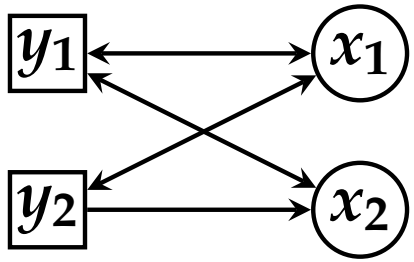
- Positional Determinacy
- Algorithms for Parity Games
- Reducing Muller Games to Parity Games

6 Applications

Muller condition

- A winning condition $Win \subseteq C^\omega$ is called a Muller condition if there is a set $\mathcal{F} \subseteq 2^C$ of sets of colors such that $\alpha \in Win$ iff $\mathrm{Inf}(c(\alpha)) \in \mathcal{F}$.
- The objective for Eva in a Muller game is to make the set of colors that appear infinitely often in a play such that it belongs to $\mathcal{F}$.
- Muller conditions are Boolean combinations of Büchi conditions: they can be seen as a kind of "disjunctive normal form" for conditions on the infinity set.
- Origin of the name: David E. Muller. *Infinite Sequences and Finite Machines*. 1963

Example



Winning condition for Eva: $y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$

As Muller condition: $\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$

Remarks:

- **It suffices to consider sets in the Muller condition that can occur as infinity set (strongly connected)**
- **Eva has a winning strategy (move to y_2 if the previous x_i was x_2) but no positional winning strategy:**
  - **Always $x_1 \longrightarrow y_1$: Adam can force $\mathrm{Inf}(\alpha) = \{x_1, x_2, y_1\}$**
  - **Always $x_1 \longrightarrow y_2$: Adam can force $\mathrm{Inf}(\alpha) = \{x_1, y_2\}$**

Memory Structures

Let $\mathcal{G}$ be a game with vertex set V. A memory structure for $\mathcal{G}$ is a tuple $\mathcal{M} = (M, m_0, \delta)$ with

- **a set M of memory states,**
- **an initial memory state m_0,**
- **a memory update function $\delta : M \times V \to M$.**

$\mathcal{M}$ computes an abstraction of the initial segments of a play:

Memory Structures

Let $\mathcal{G}$ be a game with vertex set V. A memory structure for $\mathcal{G}$ is a tuple $\mathcal{M} = (M, m_0, \delta)$ with

- **a set M of memory states,**
- **an initial memory state m_0,**
- **a memory update function $\delta : M \times V \to M$.**

$\mathcal{M}$ computes an abstraction of the initial segments of a play:

v_0

m_0

Memory Structures

Let $\mathcal{G}$ be a game with vertex set V. A memory structure for $\mathcal{G}$ is a tuple $\mathcal{M} = (M, m_0, \delta)$ with

- **a set M of memory states,**
- **an initial memory state m_0,**
- **a memory update function $\delta : M \times V \to M$.**

$\mathcal{M}$ computes an abstraction of the initial segments of a play:

$$v_0$$

$$m_0 \xrightarrow{\delta} m_1$$

Memory Structures

Let $\mathcal{G}$ be a game with vertex set V. A memory structure for $\mathcal{G}$ is a tuple $\mathcal{M} = (M, m_0, \delta)$ with

- **a set M of memory states,**
- **an initial memory state m_0,**
- **a memory update function $\delta : M \times V \to M$.**

$\mathcal{M}$ computes an abstraction of the initial segments of a play:

$$\begin{array}{ccc} & v_0 & v_1 \\ m_0 & \xrightarrow{\delta} & m_1 \end{array}$$

Memory Structures

Let $\mathcal{G}$ be a game with vertex set V. A memory structure for $\mathcal{G}$ is a tuple $\mathcal{M} = (M, m_0, \delta)$ with

- **a set M of memory states,**
- **an initial memory state m_0,**
- **a memory update function $\delta : M \times V \longrightarrow M$.**

$\mathcal{M}$ computes an abstraction of the initial segments of a play:

$$m_0 \xrightarrow[\delta]{v_0} m_1 \xrightarrow[\delta]{v_1} m_2$$

Memory Structures

Let $\mathcal{G}$ be a game with vertex set V. A memory structure for $\mathcal{G}$ is a tuple $\mathcal{M} = (M, m_0, \delta)$ with

- **a set M of memory states,**
- **an initial memory state m_0,**
- **a memory update function $\delta : M \times V \longrightarrow M$.**

$\mathcal{M}$ computes an abstraction of the initial segments of a play:

$$\begin{array}{ccccccc} & & v_0 & & v_1 & & v_2 \\ m_0 & \xrightarrow[\delta]{} & m_1 & \xrightarrow[\delta]{} & m_2 & & \end{array}$$

Memory Structures

Let $\mathcal{G}$ be a game with vertex set V. A memory structure for $\mathcal{G}$ is a tuple $\mathcal{M} = (M, m_0, \delta)$ with

- **a set M of memory states,**
- **an initial memory state m_0,**
- **a memory update function $\delta : M \times V \longrightarrow M$.**

$\mathcal{M}$ computes an abstraction of the initial segments of a play:

$$\begin{array}{ccccccc} & & v_0 & & v_1 & & v_2 \\ m_0 & \xrightarrow[\delta]{} & m_1 & \xrightarrow[\delta]{} & m_2 & \xrightarrow[\delta]{} & m_3 \end{array}$$

Memory Structures

Let $\mathcal{G}$ be a game with vertex set V. A memory structure for $\mathcal{G}$ is a tuple $\mathcal{M} = (M, m_0, \delta)$ with

- **a set M of memory states,**
- **an initial memory state m_0,**
- **a memory update function $\delta : M \times V \longrightarrow M$.**

$\mathcal{M}$ computes an abstraction of the initial segments of a play:

$$\begin{array}{ccccccccccc} & & v_0 & & v_1 & & v_2 & & v_3 & & \\ & & & & & & & & & & \ldots \\ m_0 & \xrightarrow[\delta]{} & m_1 & \xrightarrow[\delta]{} & m_2 & \xrightarrow[\delta]{} & m_3 & & & & \end{array}$$

Memory Structures

Let $\mathcal{G}$ be a game with vertex set V. A memory structure for $\mathcal{G}$ is a tuple $\mathcal{M} = (M, m_0, \delta)$ with

- a set M of memory states,
- an initial memory state m_0,
- a memory update function $\delta : M \times V \to M$.

$\mathcal{M}$ computes an abstraction of the initial segments of a play:

$$\begin{array}{ccccccccc} & & v_0 & & v_1 & & v_2 & & v_3 \\ m_0 & \xrightarrow[\delta]{} & m_1 & \xrightarrow[\delta]{} & m_2 & \xrightarrow[\delta]{} & m_3 & & \cdots \end{array}$$

Extend δ to sequences of vertices $\delta^*(V^*) \to M$:

$$\delta^*(v_0 \cdots v_i) = \begin{cases} m_0 \text{ if } i < 0 \text{ (the sequence is empty)} \\ \delta(\delta^*(v_0 \ldots v_{i-1}), v_i) \text{ otherwise} \end{cases}$$

Finite Memory Strategies

A finite memory strategy for Eva in $\mathcal{G}$ is a pair $(\mathcal{M}, f)$ of a finite memory structure $\mathcal{M} = (M, m_0, \delta)$ and a function

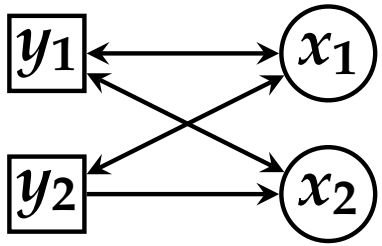
$$f : M \times V_\exists \longrightarrow V$$

such that $f(m, v) = v'$ implies $(v, v') \in E$.

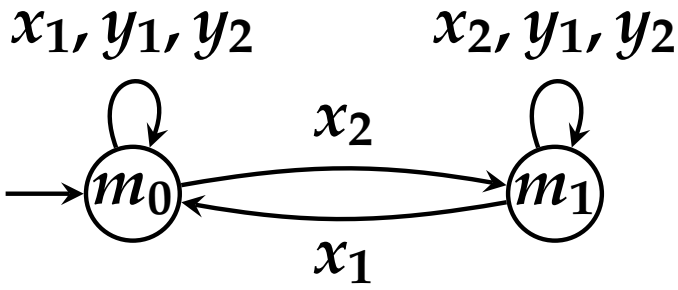
The strategy $f_\mathcal{M}$ defined by $(\mathcal{M}, f)$ is given by

$$f_\mathcal{M}(v_0 \cdots v_i) = f(\delta^*(v_0 \cdots v_{i-1}), v_i)$$

Example



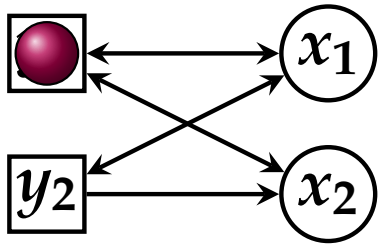
$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$



$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

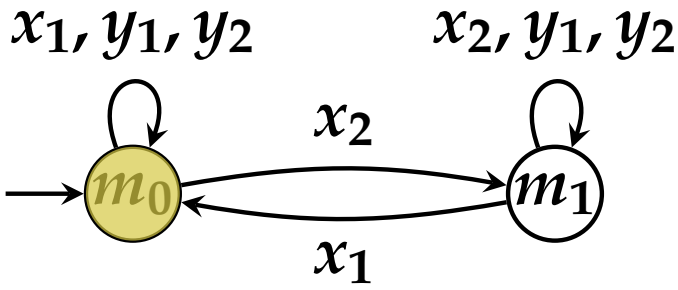
Example



$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$

y_1

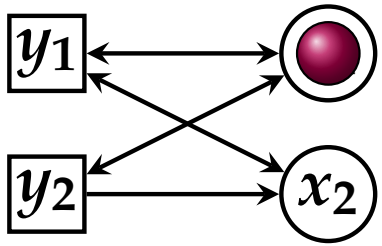
m_0



$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

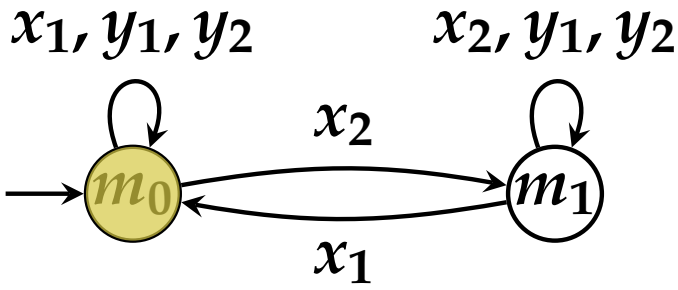
"Move to y_2 if the previous x_i was x_2"

Example



$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$

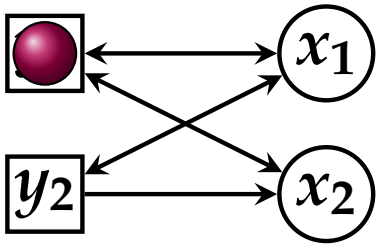
y_1 x_1
m_0 m_0



$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

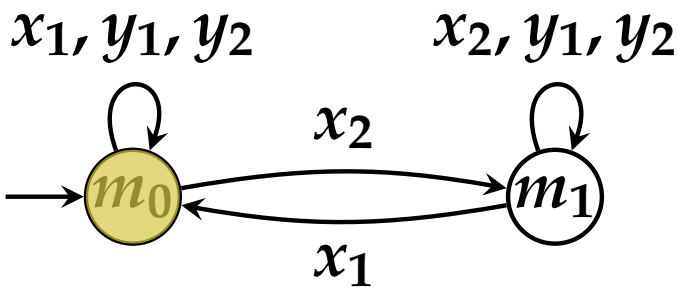
"Move to y_2 if the previous x_i was x_2"

Example



$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$

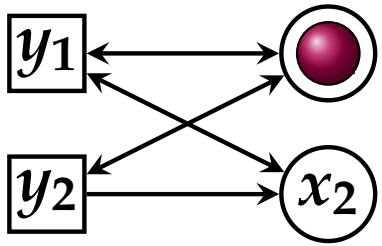
$$\begin{array}{ccc} y_1 & x_1 & y_1 \\ m_0 & m_0 & m_0 \end{array}$$



$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

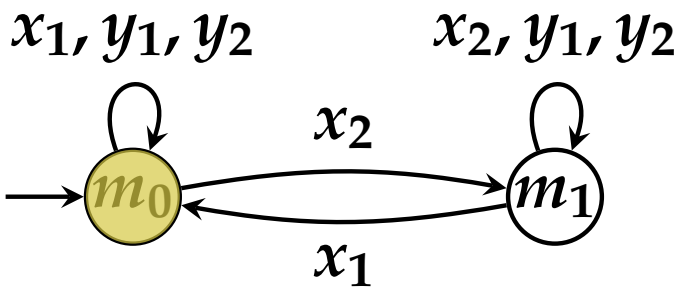
"Move to y_2 if the previous x_i was x_2"

Example



$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$

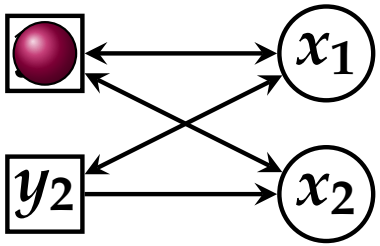
$$\begin{array}{cccc} y_1 & x_1 & y_1 & x_1 \\ m_0 & m_0 & m_0 & m_0 \end{array}$$



$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

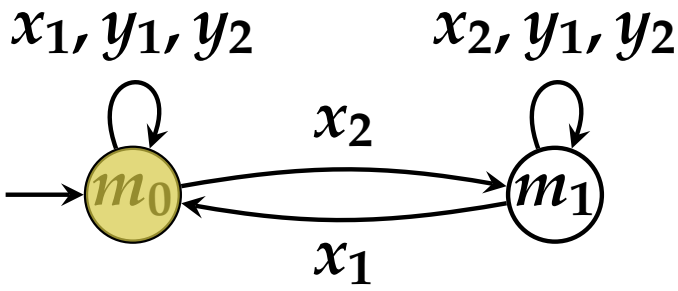
"Move to y_2 if the previous x_i was x_2"

Example



$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$

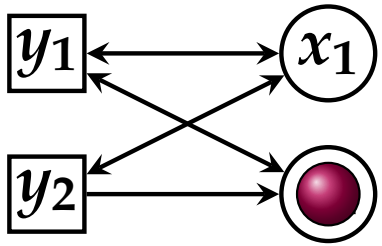
$$\begin{array}{ccccc} y_1 & x_1 & y_1 & x_1 & y_1 \\ m_0 & m_0 & m_0 & m_0 & m_0 \end{array}$$



$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

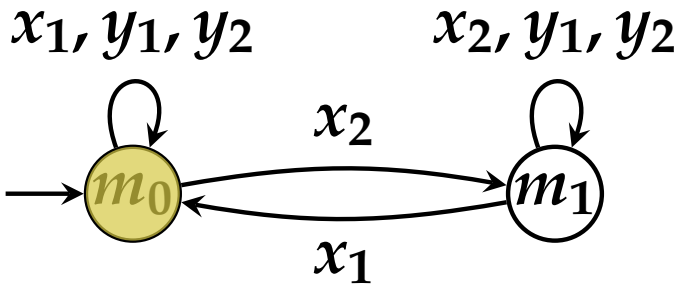
"Move to y_2 if the previous x_i was x_2"

Example



$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$

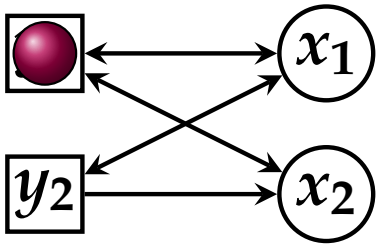
$$\begin{array}{cccccc} y_1 & x_1 & y_1 & x_1 & y_1 & x_2 \\ m_0 & m_0 & m_0 & m_0 & m_0 & m_0 \end{array}$$



$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

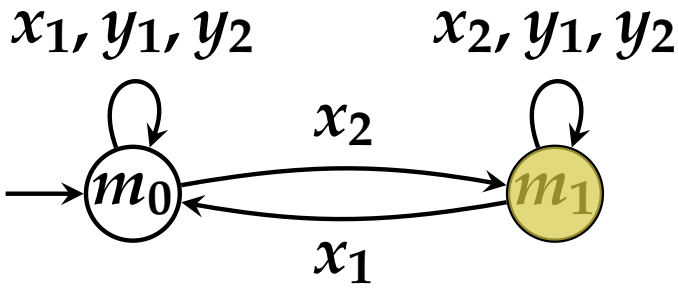
"Move to y_2 if the previous x_i was x_2"

Example



$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$

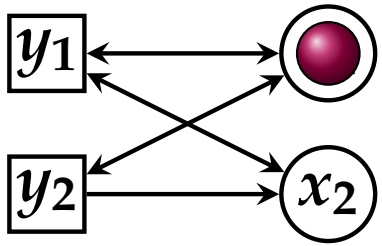
$$\begin{array}{ccccccc} y_1 & x_1 & y_1 & x_1 & y_1 & x_2 & y_1 \\ m_0 & m_0 & m_0 & m_0 & m_0 & m_0 & m_1 \end{array}$$



$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

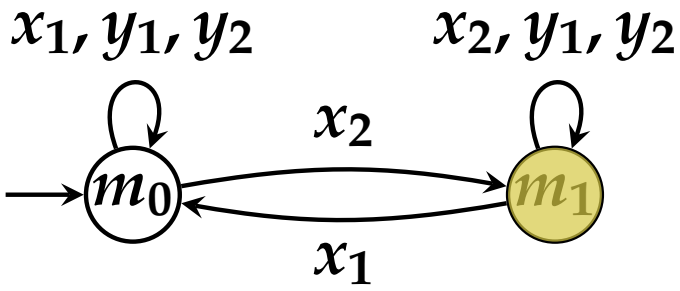
"Move to y_2 if the previous x_i was x_2"

Example



$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$

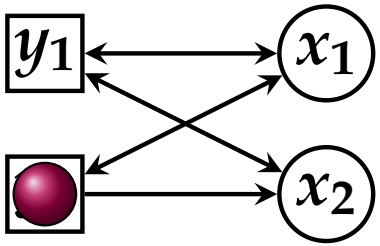
y_1	x_1	y_1	x_1	y_1	x_2	y_1	x_1
m_0	m_0	m_0	m_0	m_0	m_0	m_1	m_1



$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

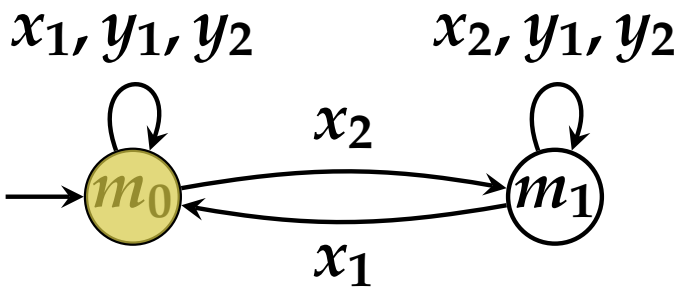
"Move to y_2 if the previous x_i was x_2"

Example



$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$

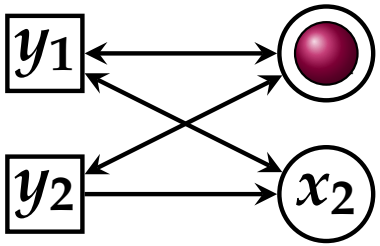
y_1	x_1	y_1	x_1	y_1	x_2	y_1	x_1	y_2
m_0	m_0	m_0	m_0	m_0	m_0	m_1	m_1	m_0



$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

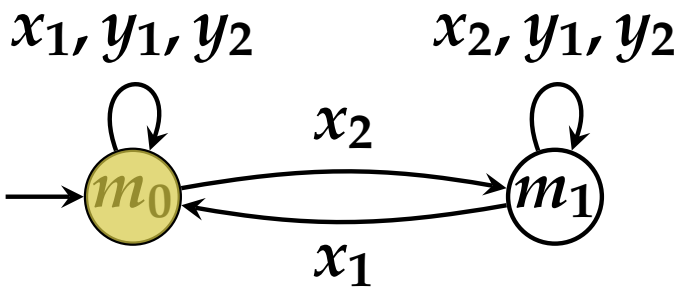
"Move to y_2 if the previous x_i was x_2"

Example



$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$

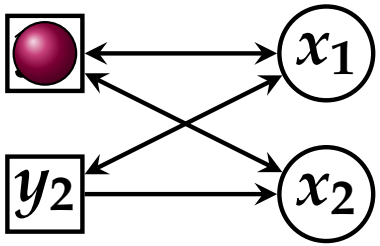
y_1	x_1	y_1	x_1	y_1	x_2	y_1	x_1	y_2	x_1
m_0	m_0	m_0	m_0	m_0	m_0	m_1	m_1	m_0	m_0



$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

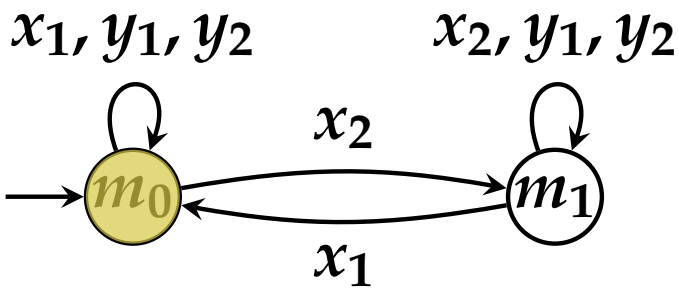
"Move to y_2 if the previous x_i was x_2"

Example



$$\mathcal{F} = \{\{x_1, y_1\}, \{x_2, y_1\}, \{x_1, x_2, y_1, y_2\}\}$$

y_1	x_1	y_1	x_1	y_1	x_2	y_1	x_1	y_2	x_1	y_1
m_0	m_0	m_0	m_0	m_0	m_0	m_1	m_1	m_0	m_0	m_0



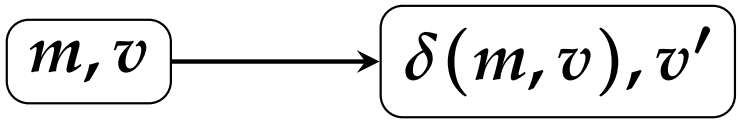
$$f(m_0, x_1) = y_1$$
$$f(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

Computing Strategies with Memory

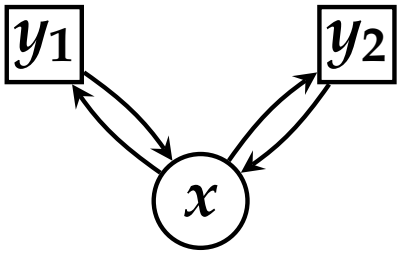
Schema of game reduction:

1. **Find a suitable memory structure $\mathcal{M}$.**
2. **Take the product of the game graph with the memory:**



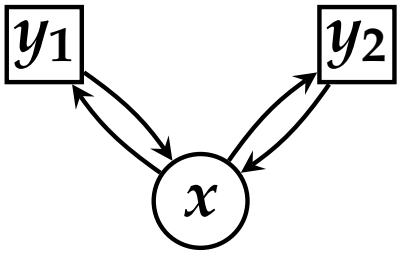
3. **On the new graph define a new winning condition that is equivalent to the original winning condition. This new winning condition should be positionally determined.**
4. **Compute positional winning strategies on this new game.**
5. **Positional strategies on the extended graph correspond to strategies with memory $\mathcal{M}$ on the original graph.**

Example



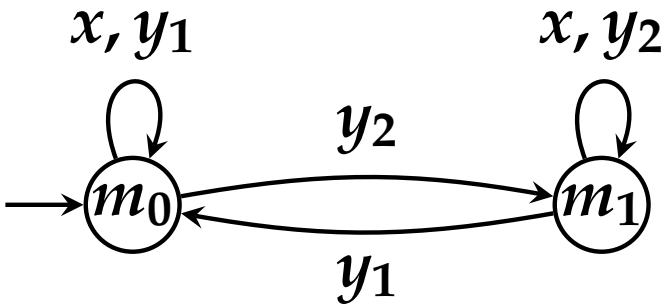
Eva has to visit y_1 and y_2 infinitely often

Example

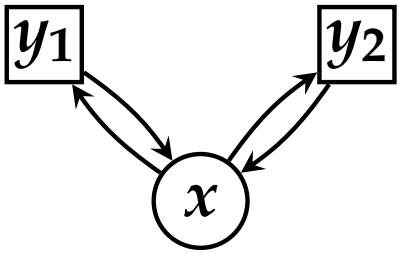


Eva has to visit y_1 and y_2 infinitely often

Memory: recall whether y_1 or y_2 was visited last:

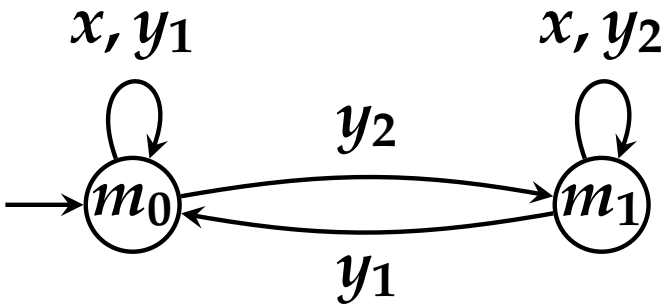


Example

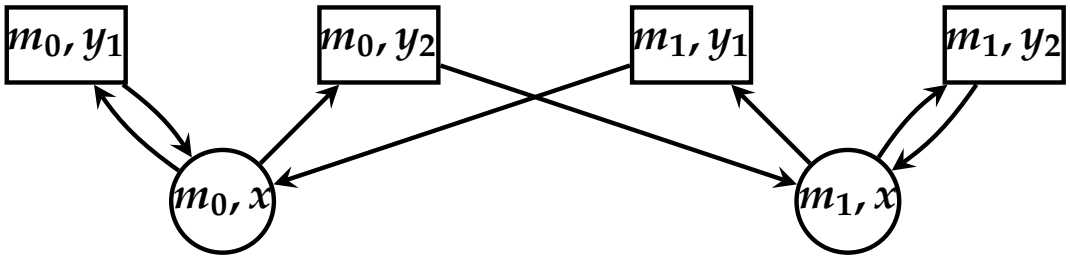


Eva has to visit y_1 and y_2 infinitely often

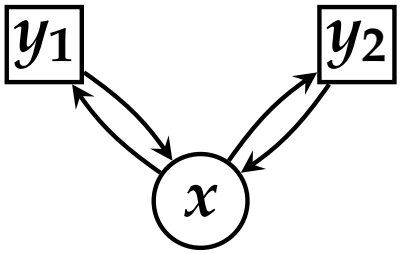
Memory: recall whether y_1 or y_2 was visited last:



Product of game and memory:

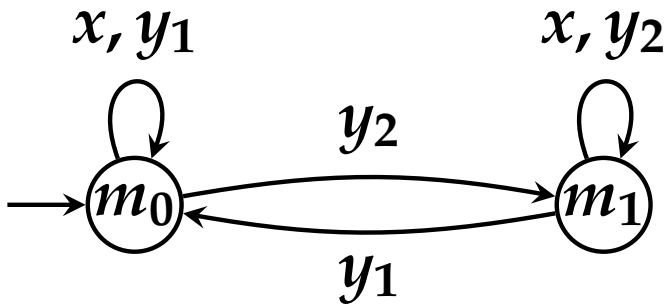


Example

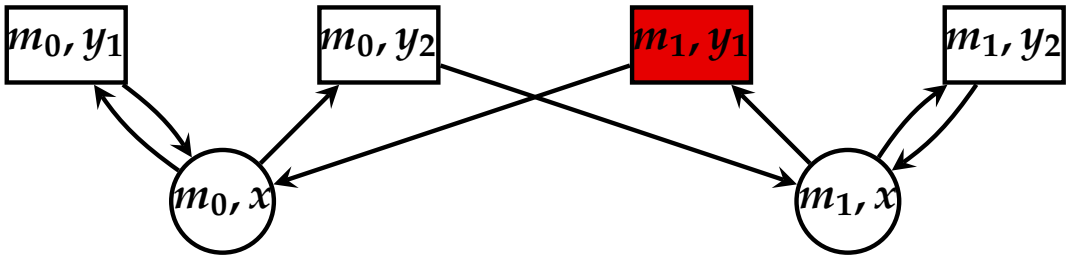


Eva has to visit y_1 and y_2 infinitely often

Memory: recall whether y_1 or y_2 was visited last:

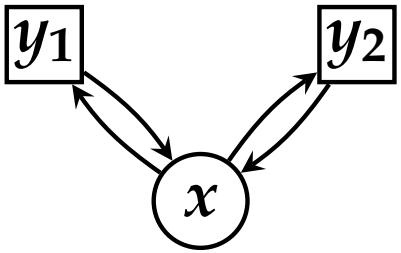


Product of game and memory:



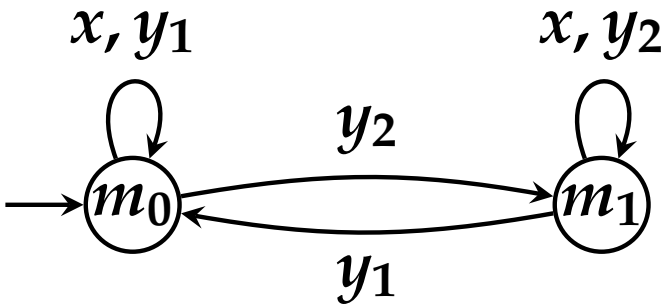
Eva has to see (m_1, y_1) infinitely often: Büchi condition!

Example

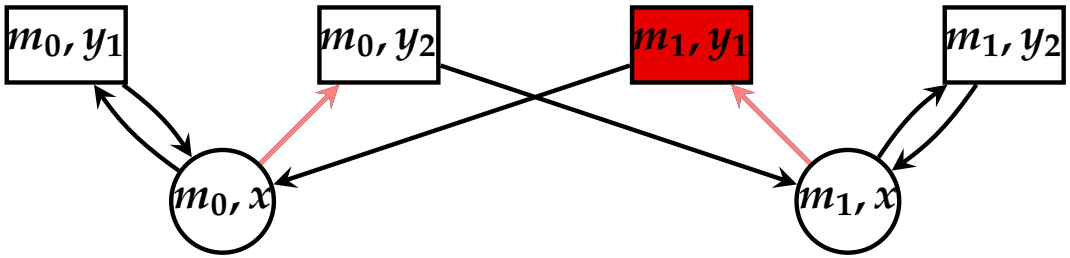


Eva has to visit y_1 and y_2 infinitely often

Memory: recall whether y_1 or y_2 was visited last:



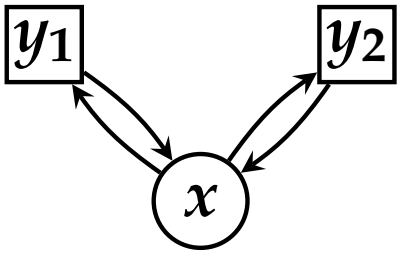
Product of game and memory:



positional strategy

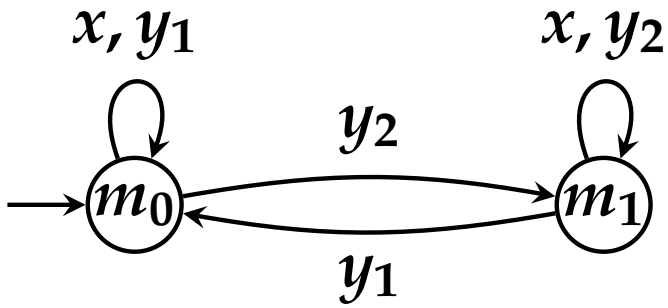
Eva has to see (m_1, y_1) infinitely often: Büchi condition!

Example



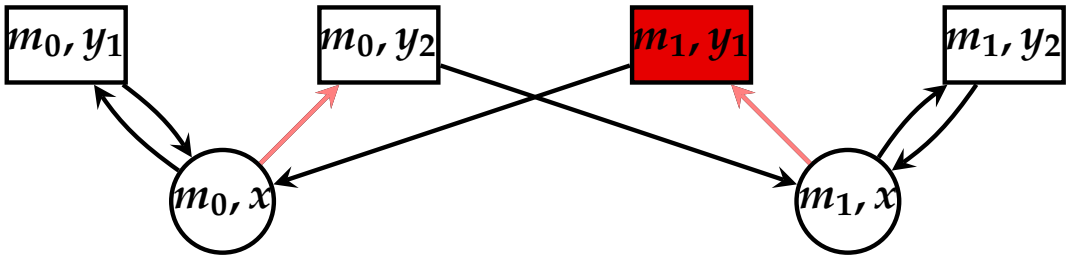
Eva has to visit y_1 and y_2 infinitely often

Memory: recall whether y_1 or y_2 was visited last:



$$f(m_0, x) = y_2$$
$$f(m_1, x) = y_1$$

Product of game and memory:



positional strategy

Eva has to see (m_1, y_1) infinitely often: Büchi condition!

Summary

- **Muller conditions (boolean combinations of Büchi conditions)**
- **Positional strategies are not enough**
- **Memory structures and strategies with memory**
- **Schema of game reduction**

1 Basics

2 Positional Strategies: Reachability and Attractors

3 Büchi Games

4 Muller Games and Finite Memory Strategies

5 Parity Games

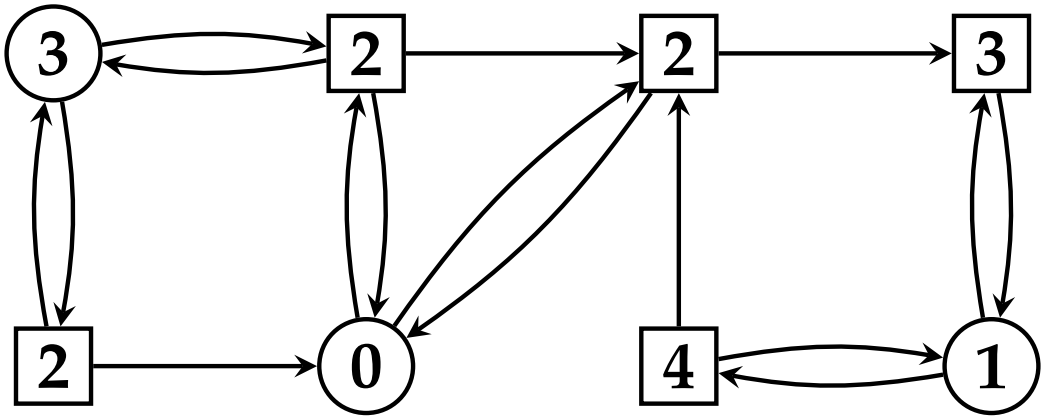
- Positional Determinacy
- Algorithms for Parity Games
- Reducing Muller Games to Parity Games

6 Applications

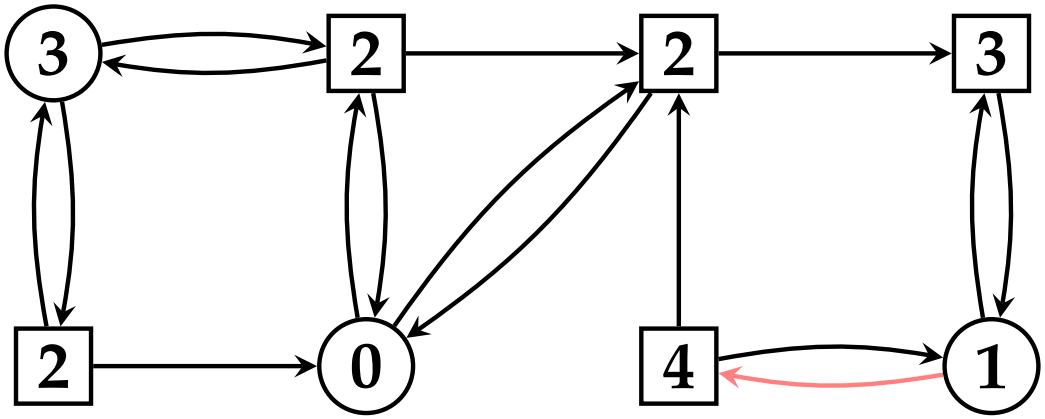
Parity Condition

- Let C be a (finite) subset of the natural numbers. A play α satisfies the (max-)parity condition if the maximal number that occurs infinitely often in $c(\alpha)$ is even.
- The colors are also called priorities in this setting.
- Sometimes parity conditions are called Rabin chain conditions.
- Büchi conditions can be expressed as parity conditions with priorities 1 and 2: map the states in F to 2 and the other ones to 1.

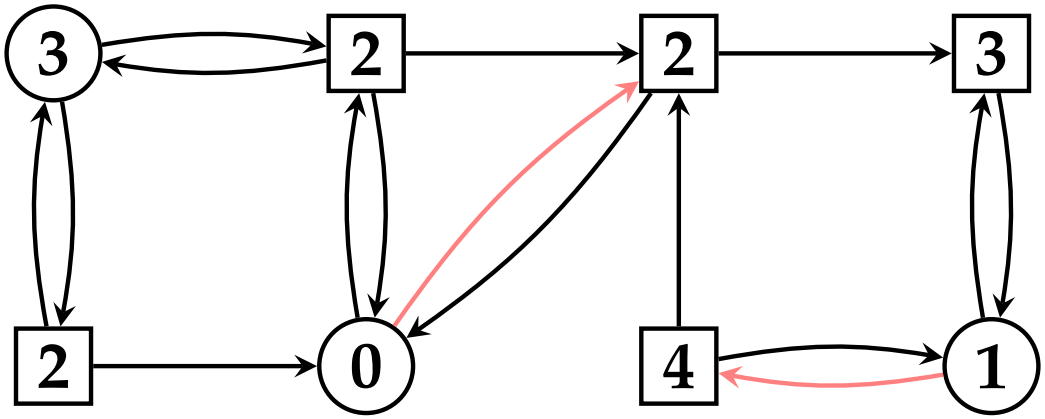
Example



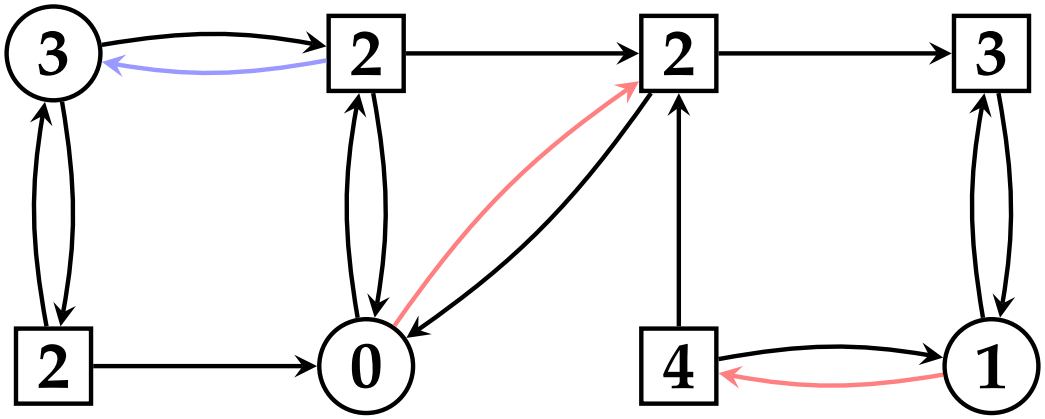
Example



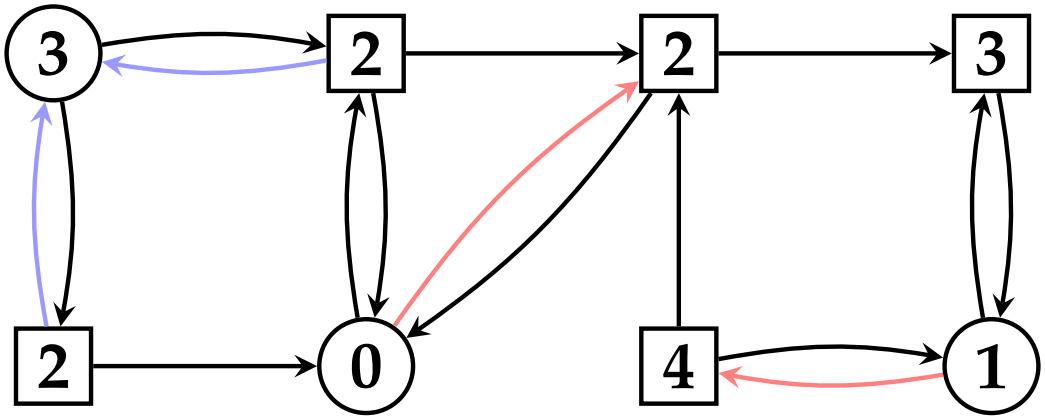
Example



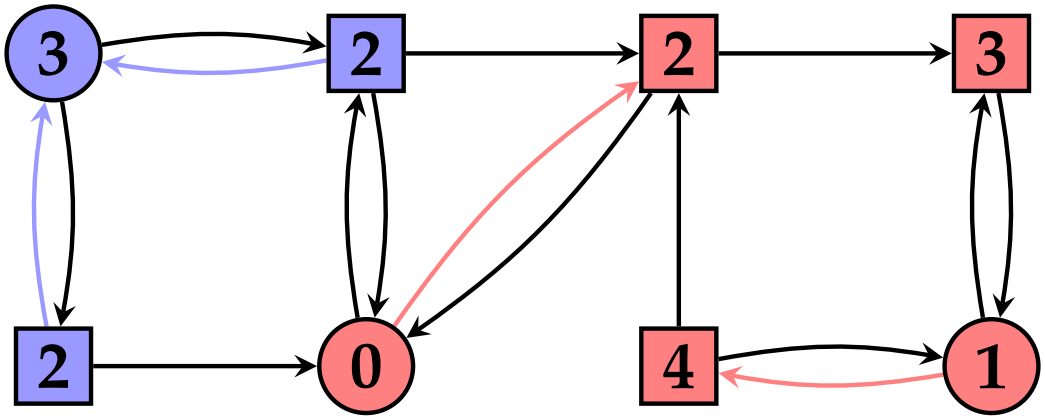
Example



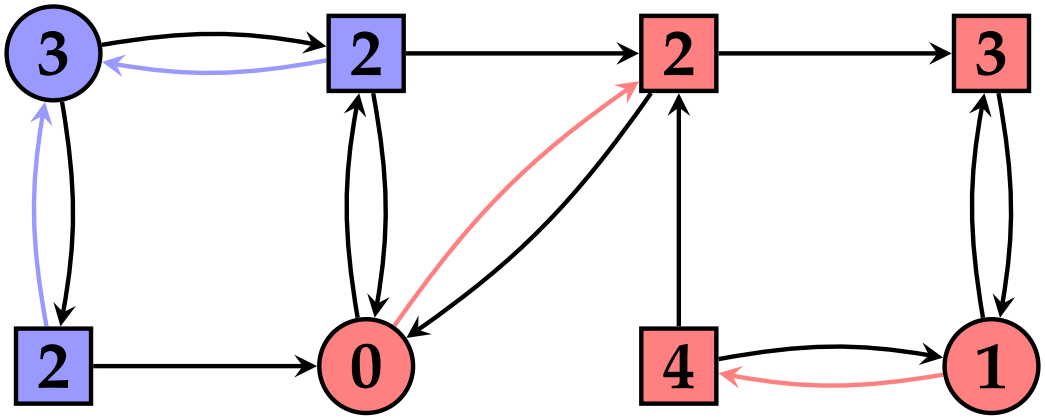
Example



Example



Example



- In this example both players have positional and uniform winning strategies on their winning area.
- For a set X of vertices a uniform winning strategy on X is a winning strategy that is winning from each vertex in X.

Positional Determinacy

Positional Determinacy of Parity Games

Theorem

Parity games are positionally determined with uniform positional winning strategies for both players on their winning areas.

Positional Determinacy of Parity Games

Theorem

Parity games are positionally determined with uniform positional winning strategies for both players on their winning areas.

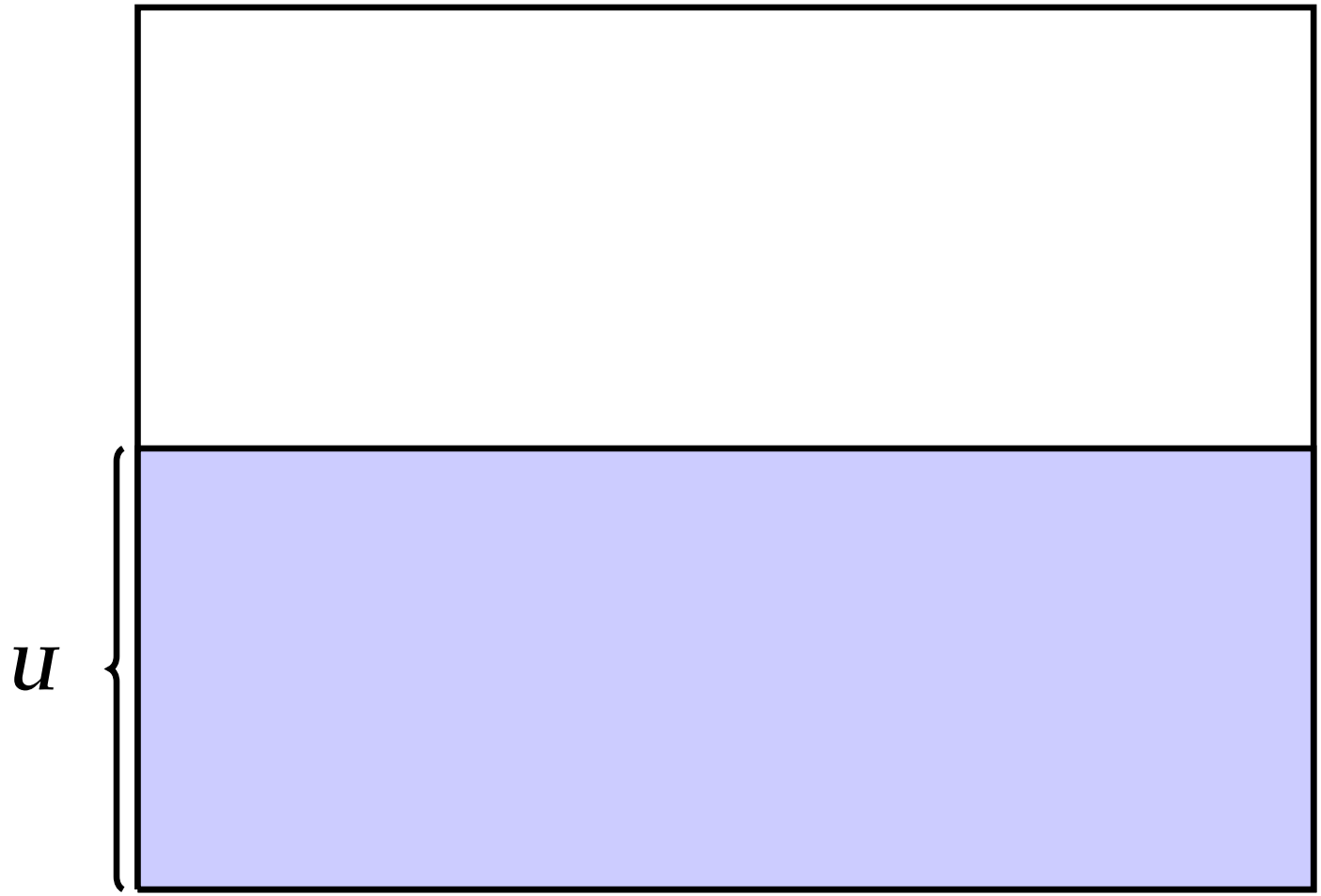
A non-constructive proof:

- **Induction on the number of priorities**
- **Let k be the maximal priority and assume that it is even**
- **Let U be the set of vertices on which Adam has a positional winning strategy**

Step 1 – Uniform Strategy

Lemma

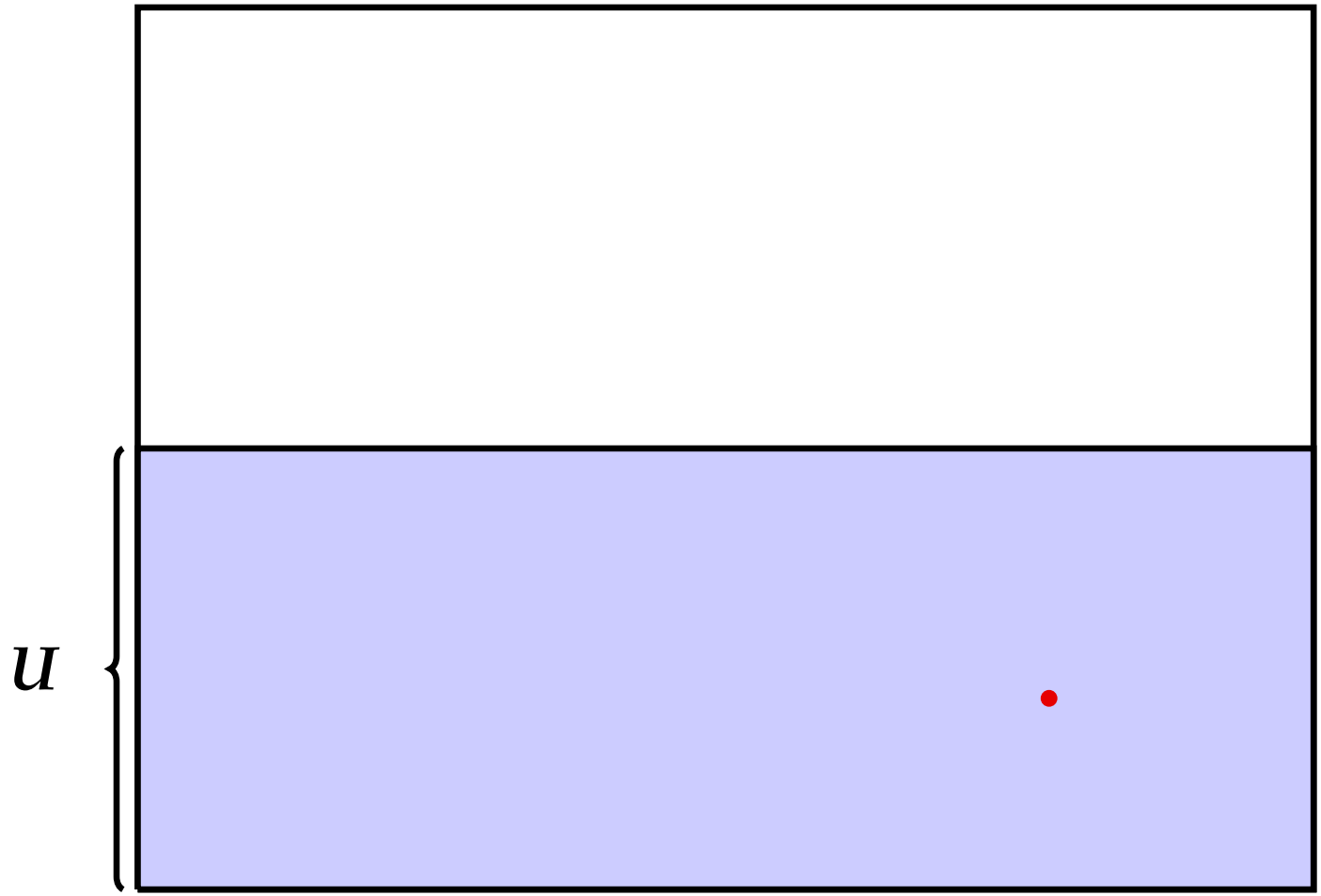
On U Adam has a uniform positional winning strategy f_U.



Step 1 – Uniform Strategy

Lemma

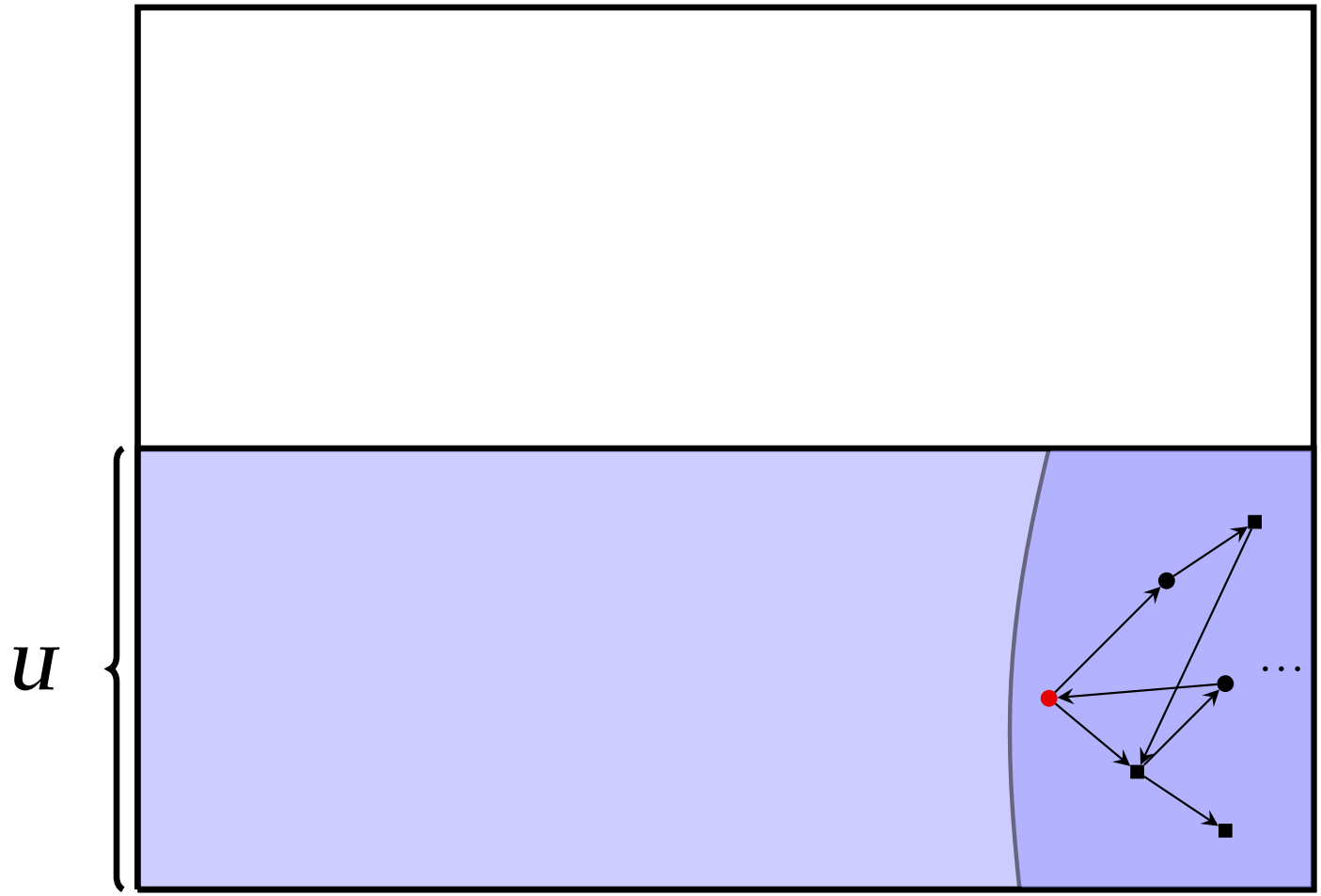
On U Adam has a uniform positional winning strategy f_U.



Step 1 – Uniform Strategy

Lemma

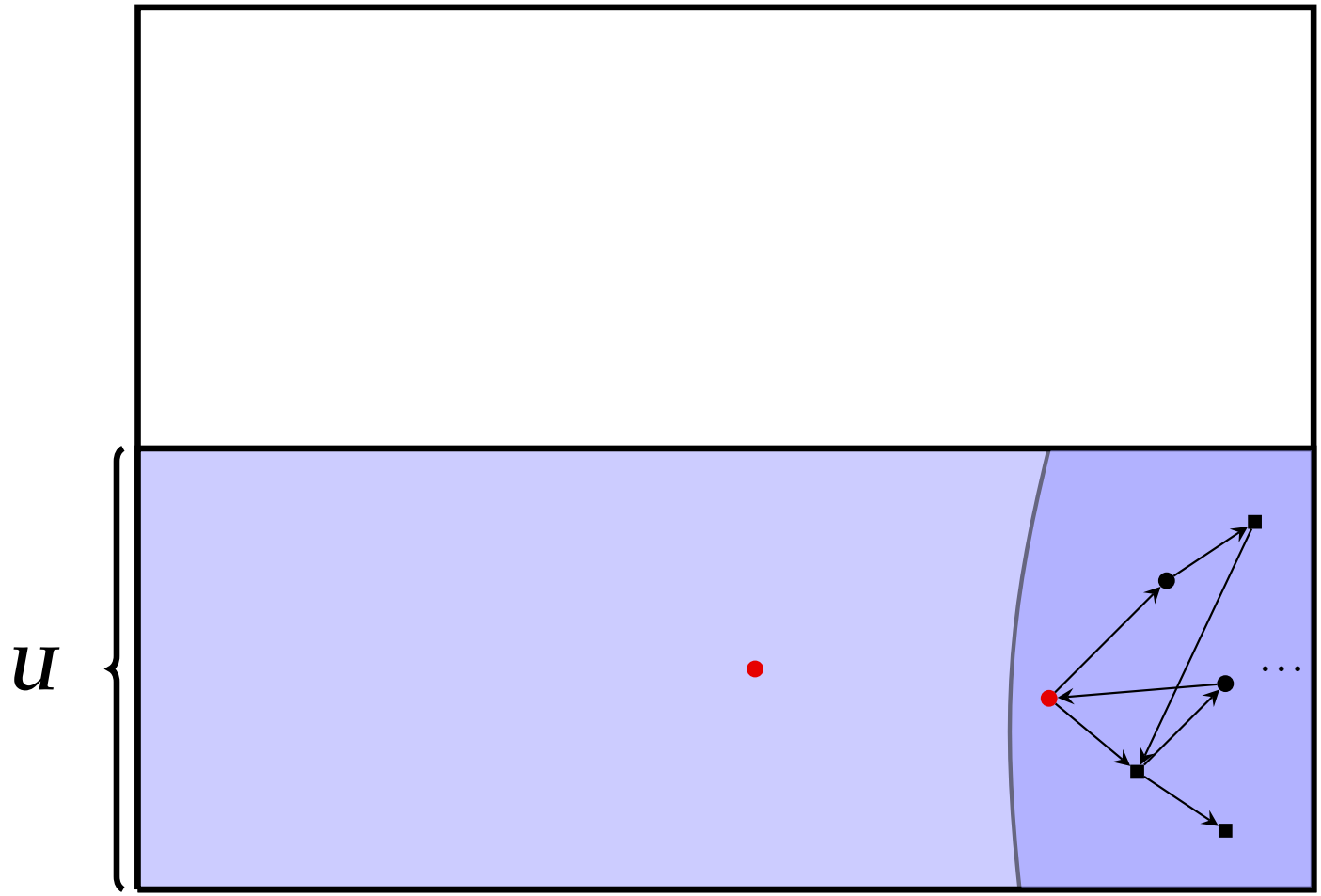
On U Adam has a uniform positional winning strategy f_U.



Step 1 – Uniform Strategy

Lemma

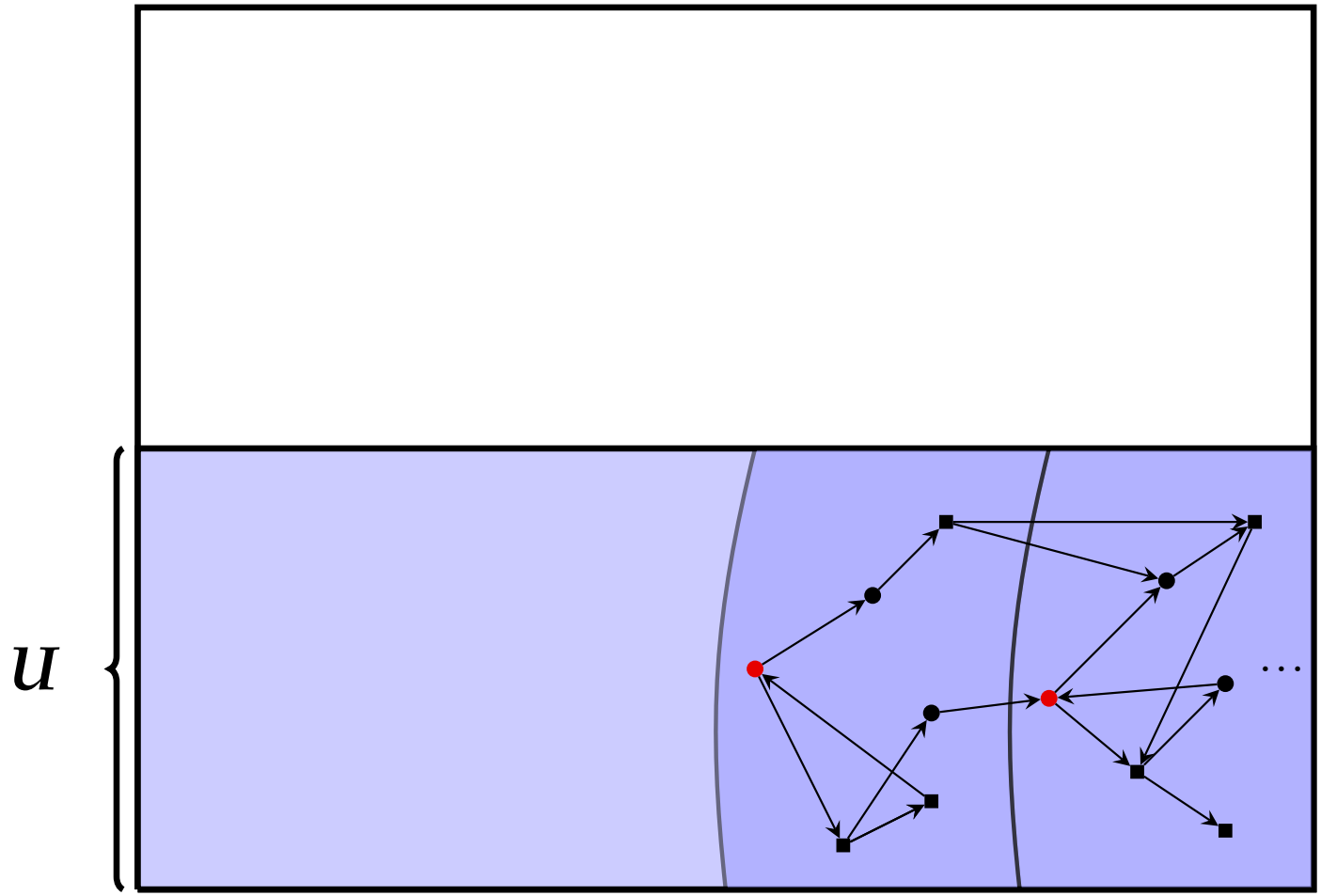
On U Adam has a uniform positional winning strategy f_U.



Step 1 – Uniform Strategy

Lemma

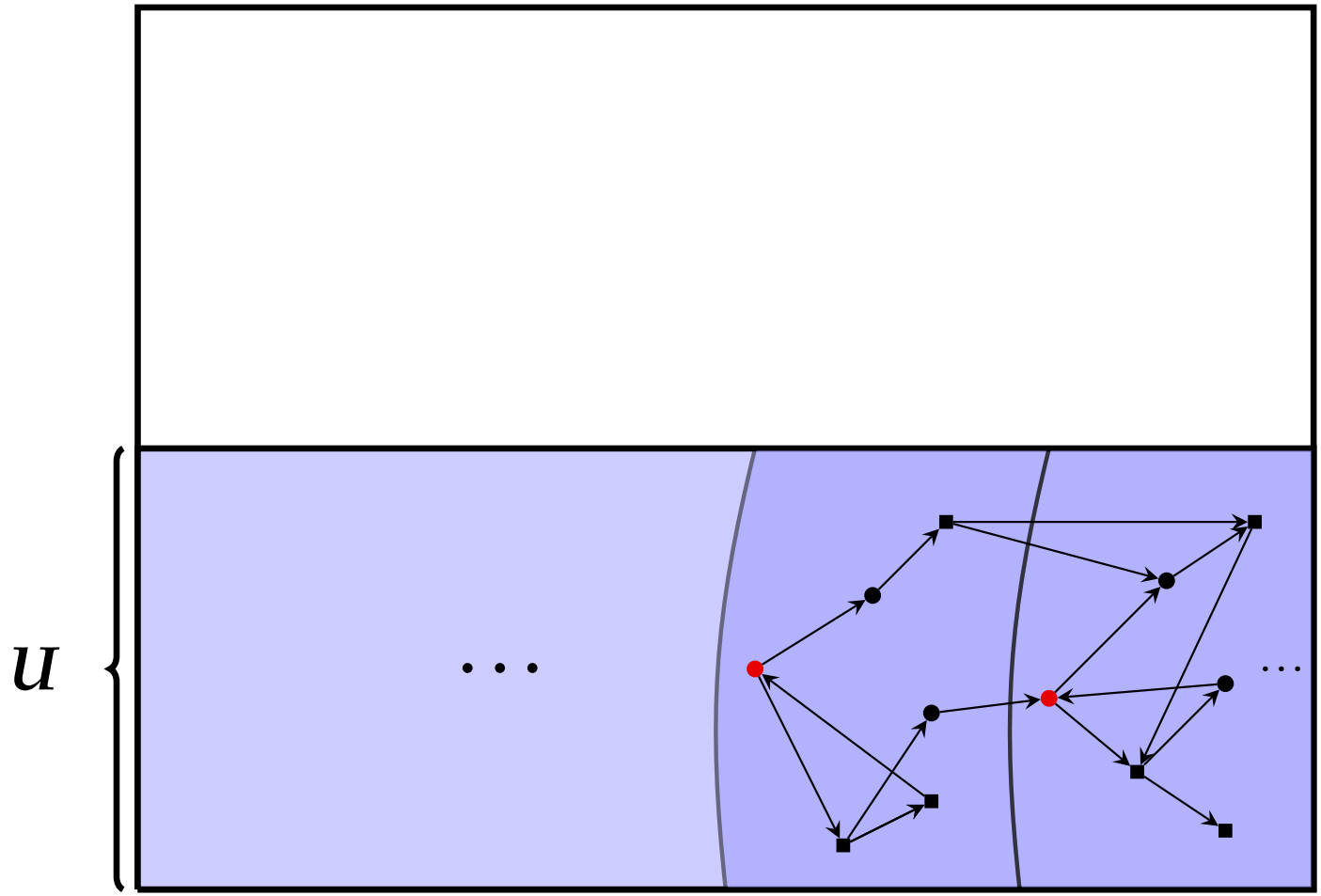
On U Adam has a uniform positional winning strategy f_U.



Step 1 – Uniform Strategy

Lemma

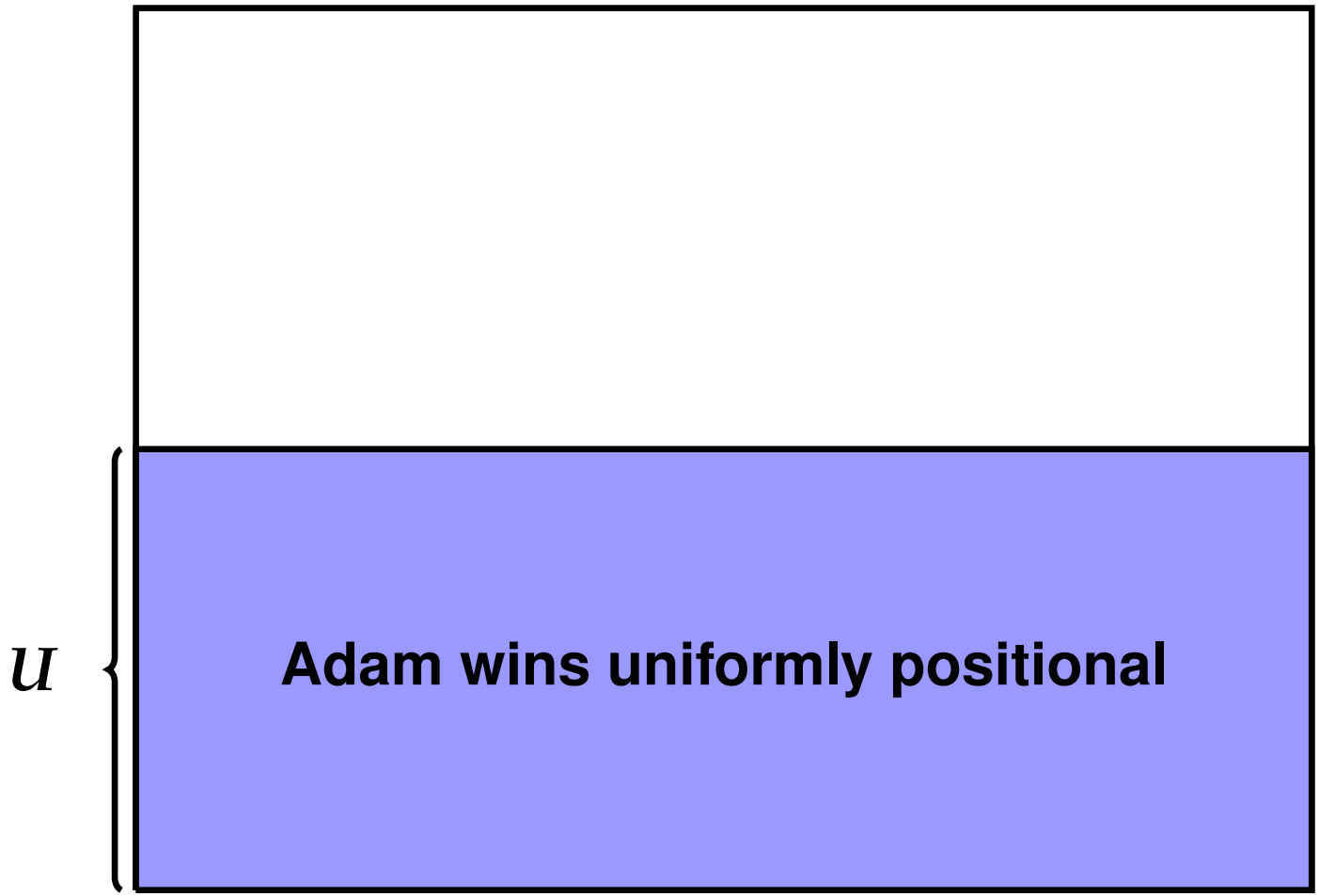
On U Adam has a uniform positional winning strategy f_U.



Step 2 – Strategy for Eva

Lemma

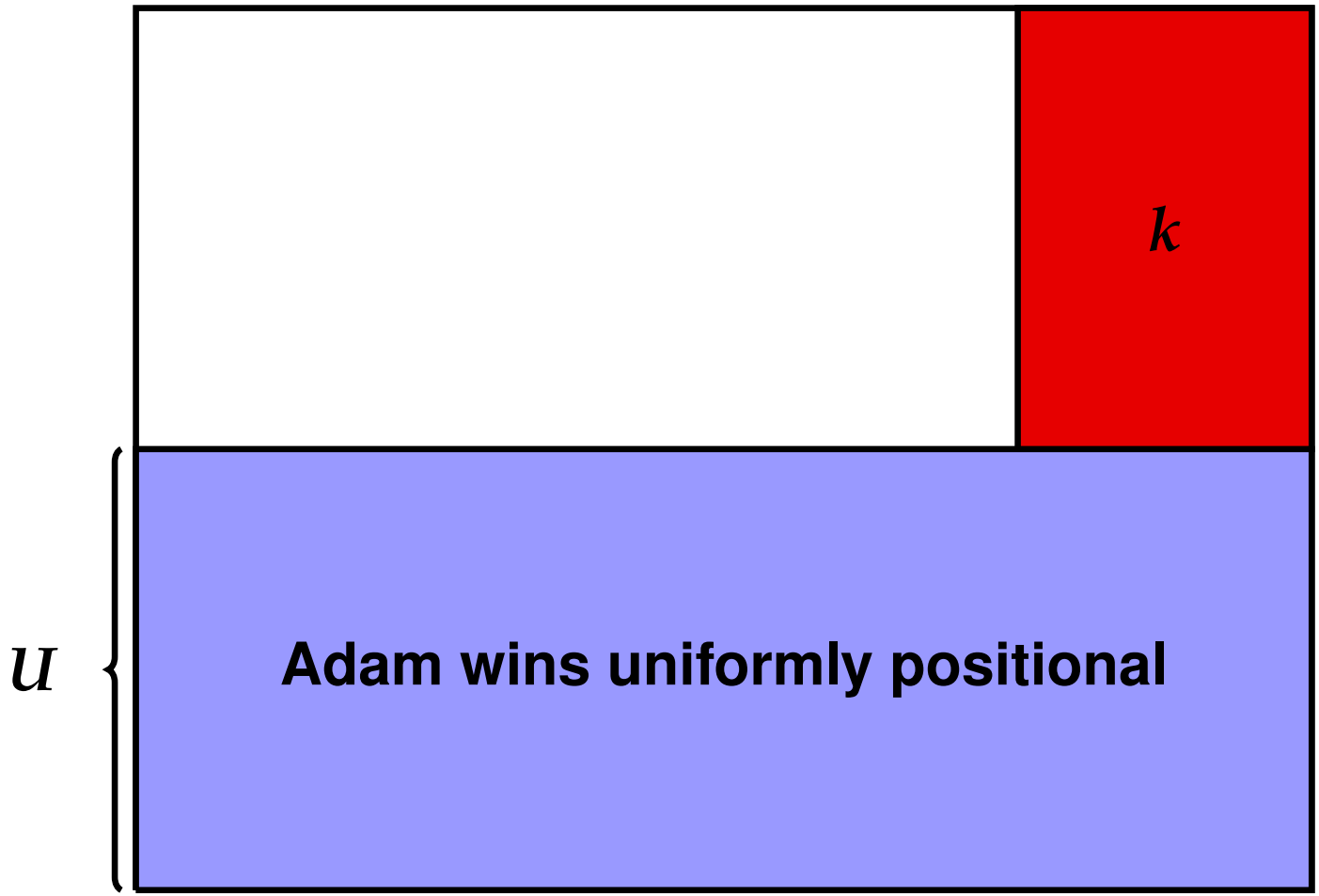
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

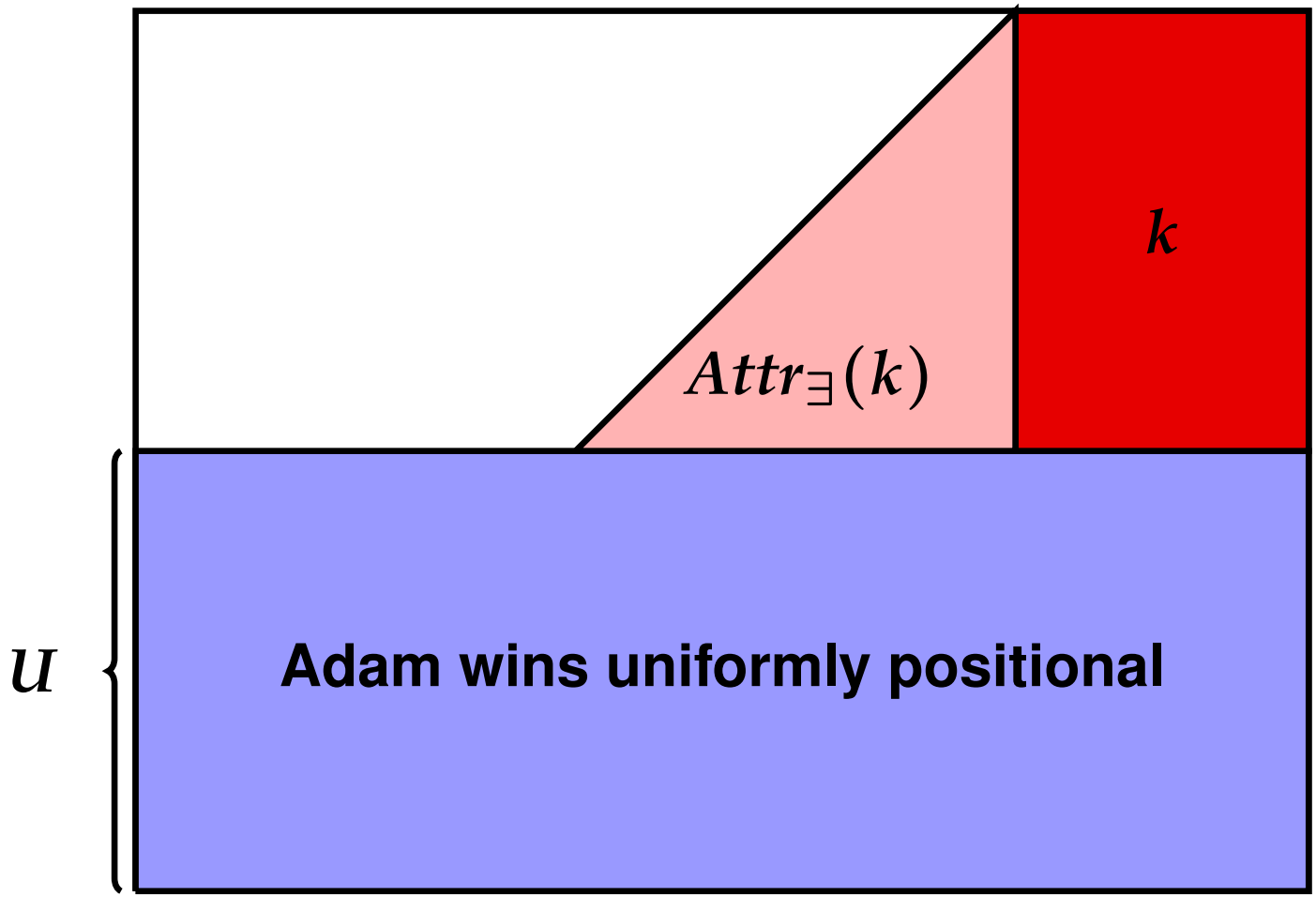
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

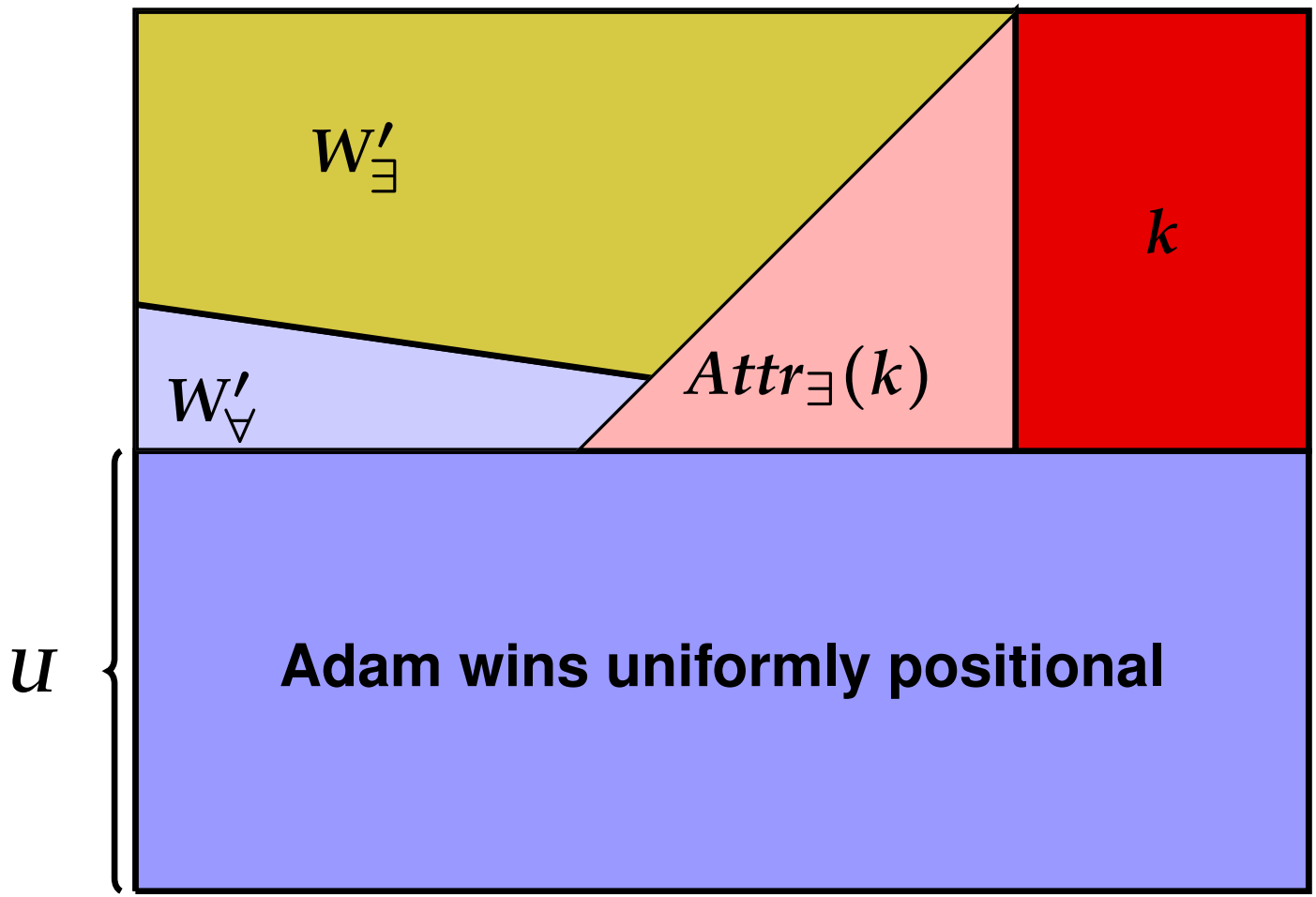
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

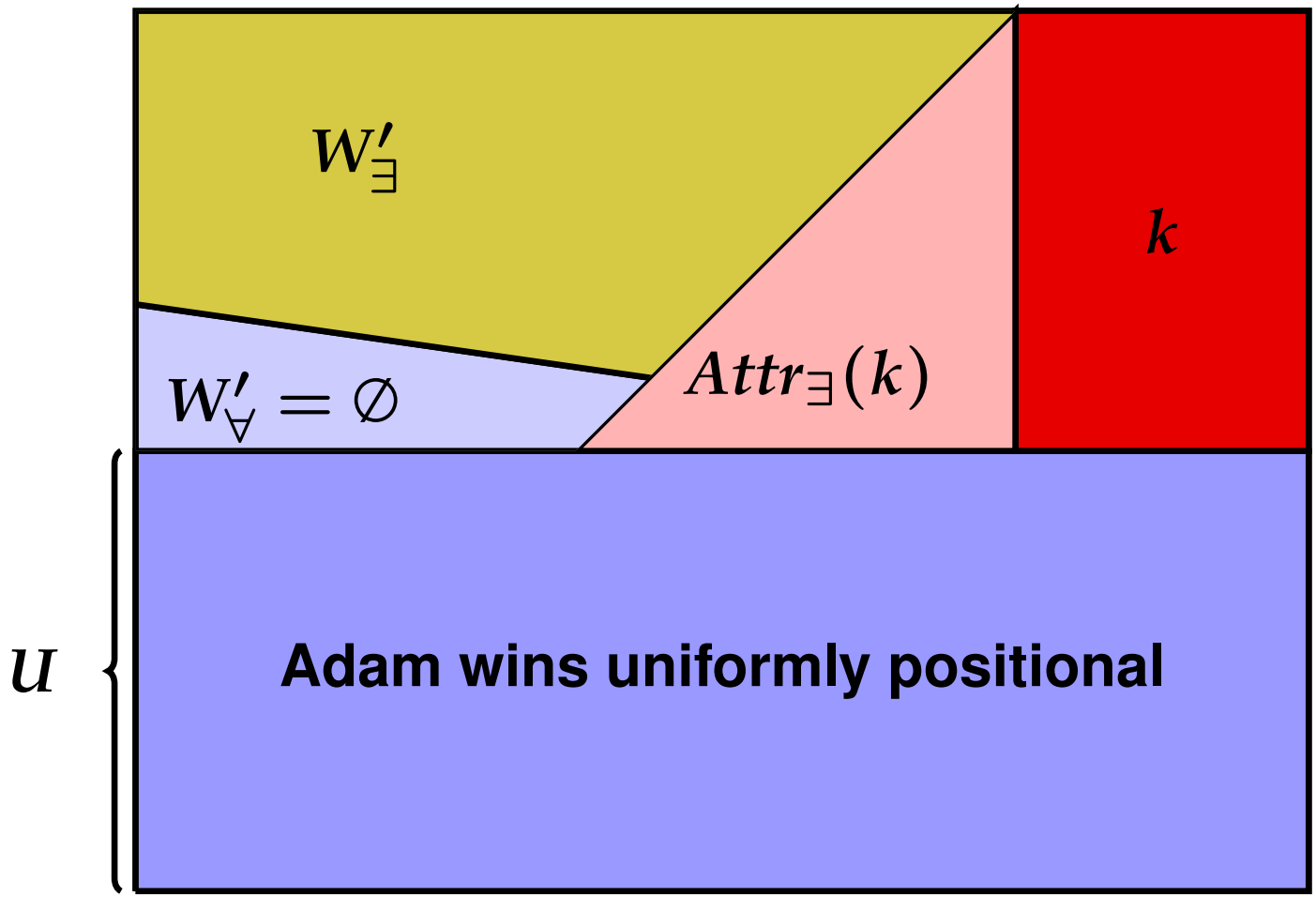
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

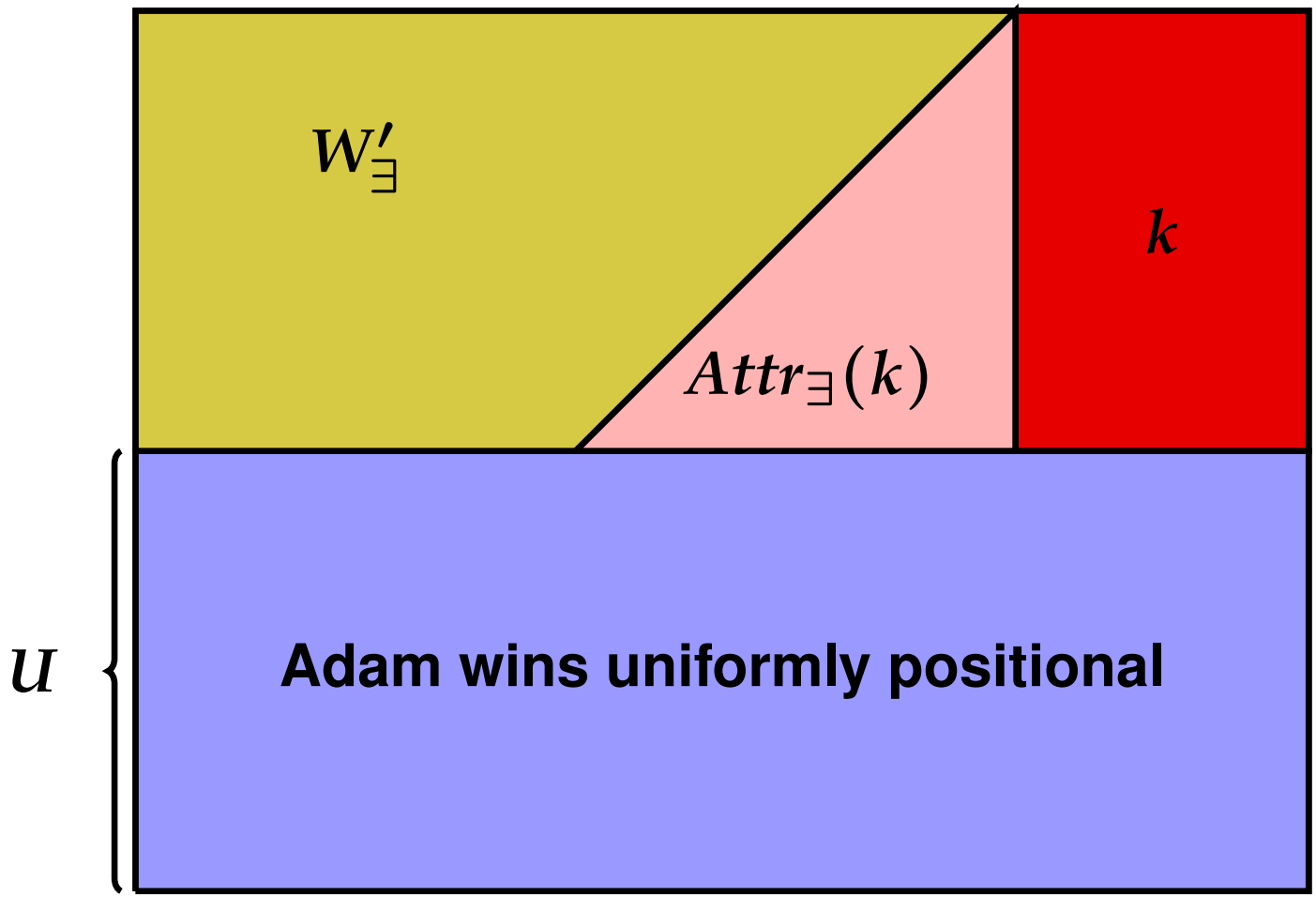
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

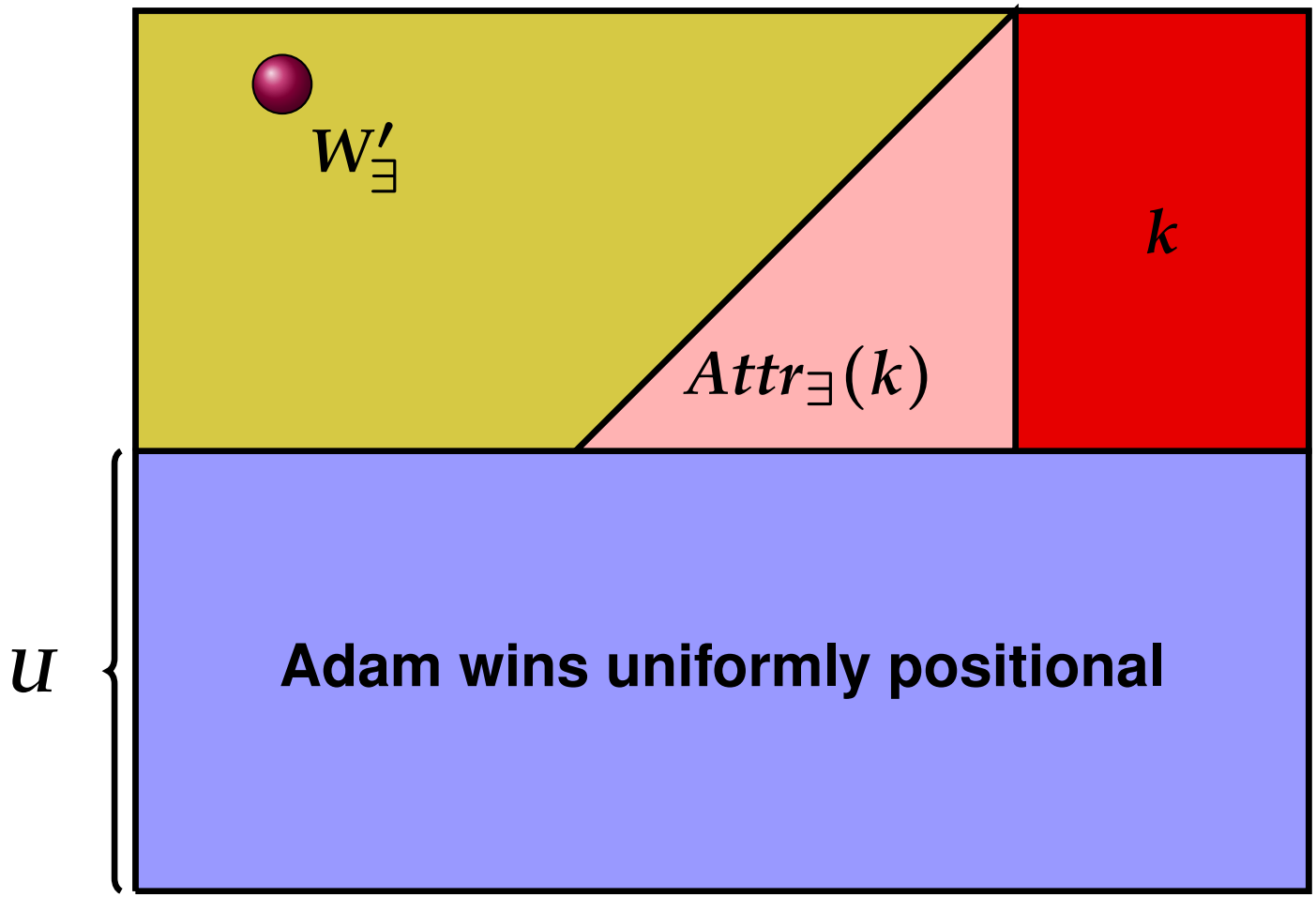
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

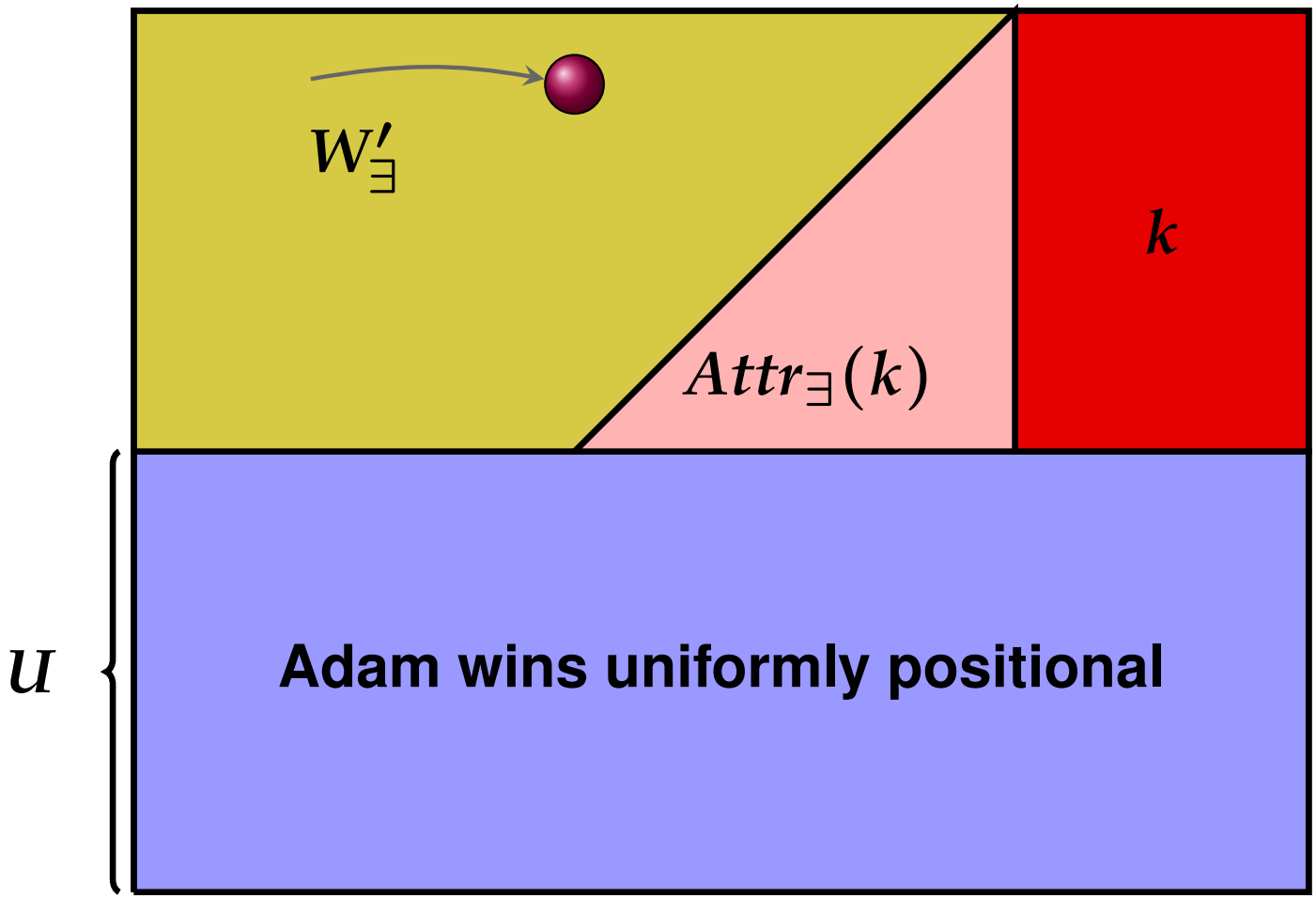
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

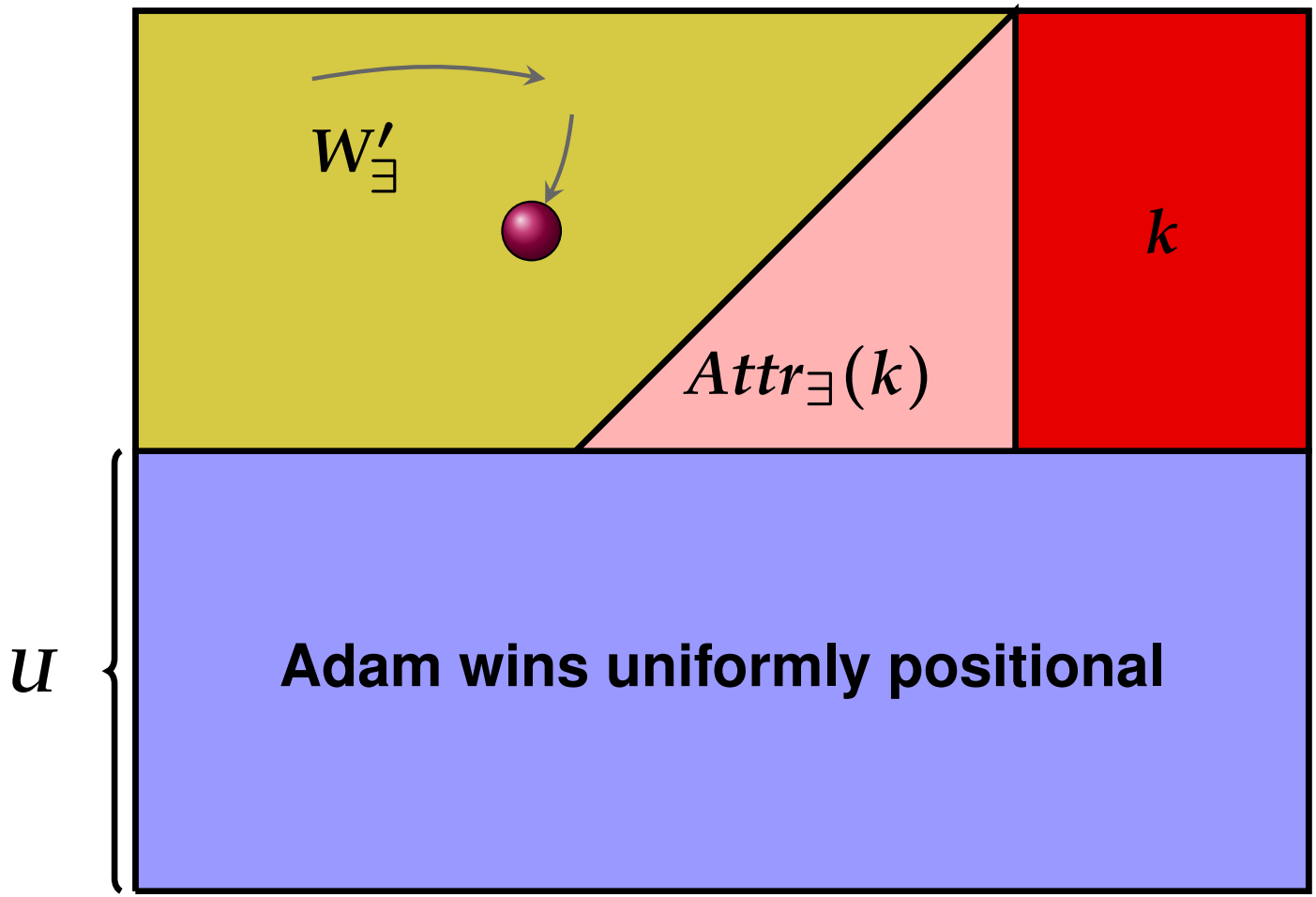
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

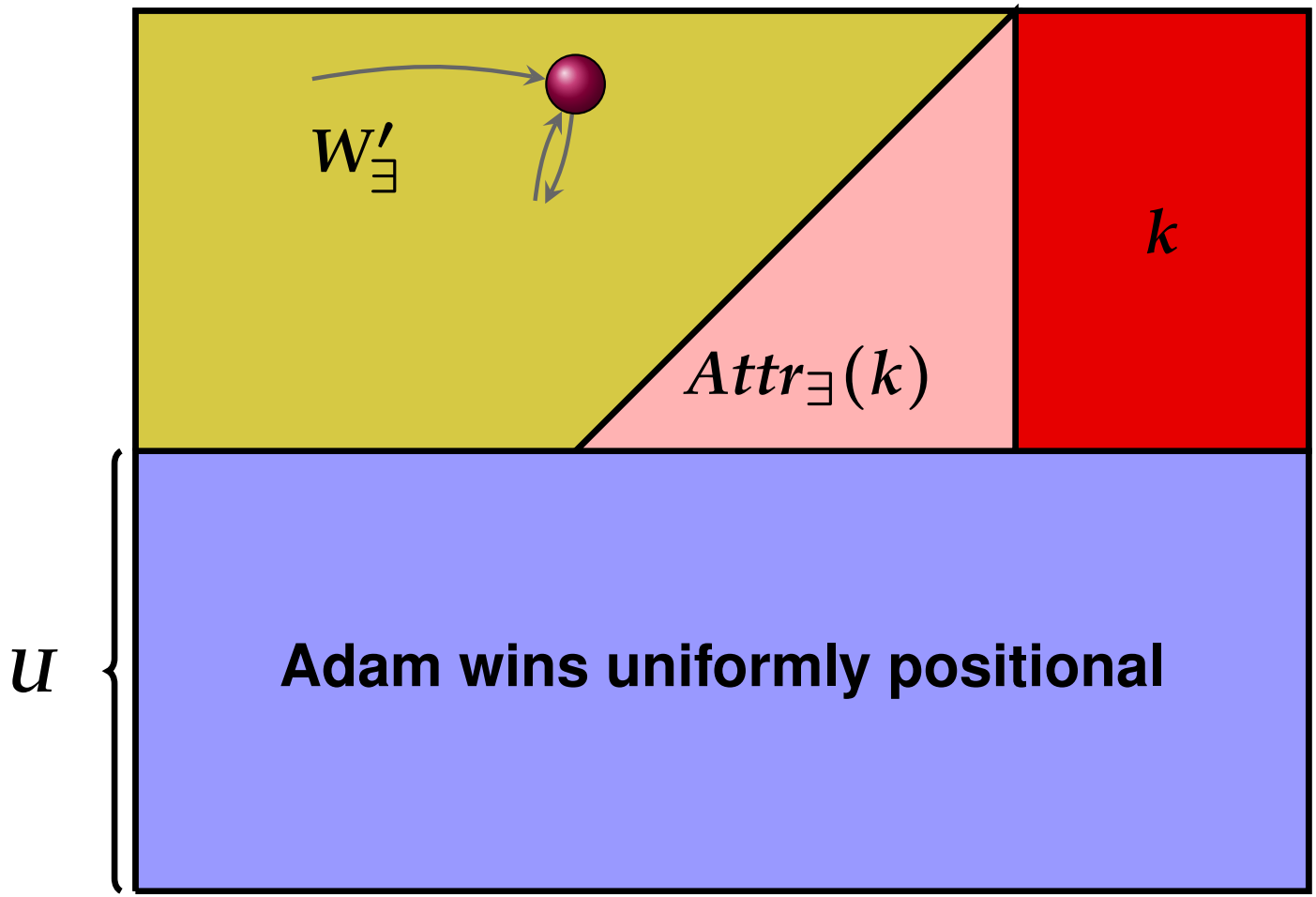
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

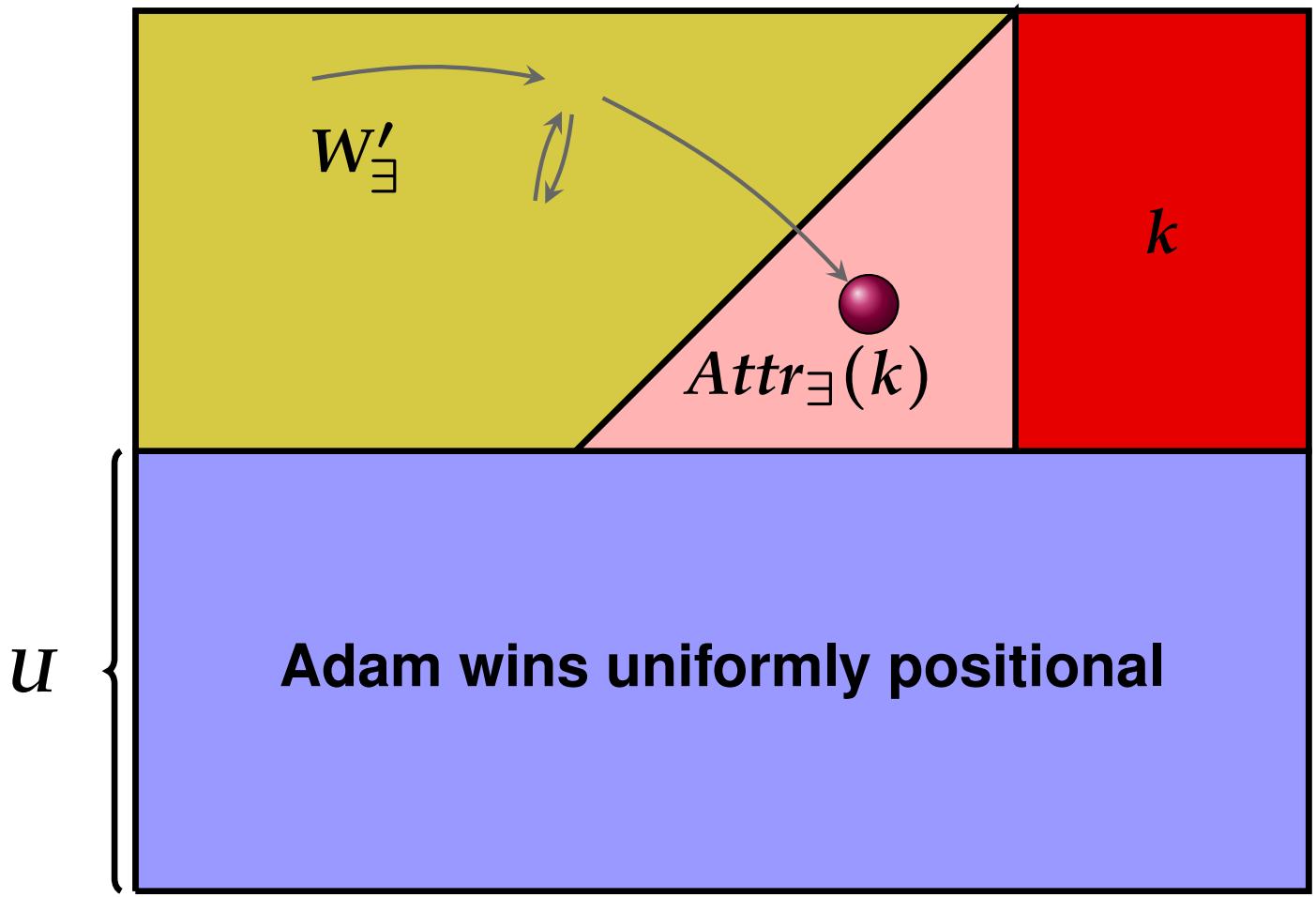
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

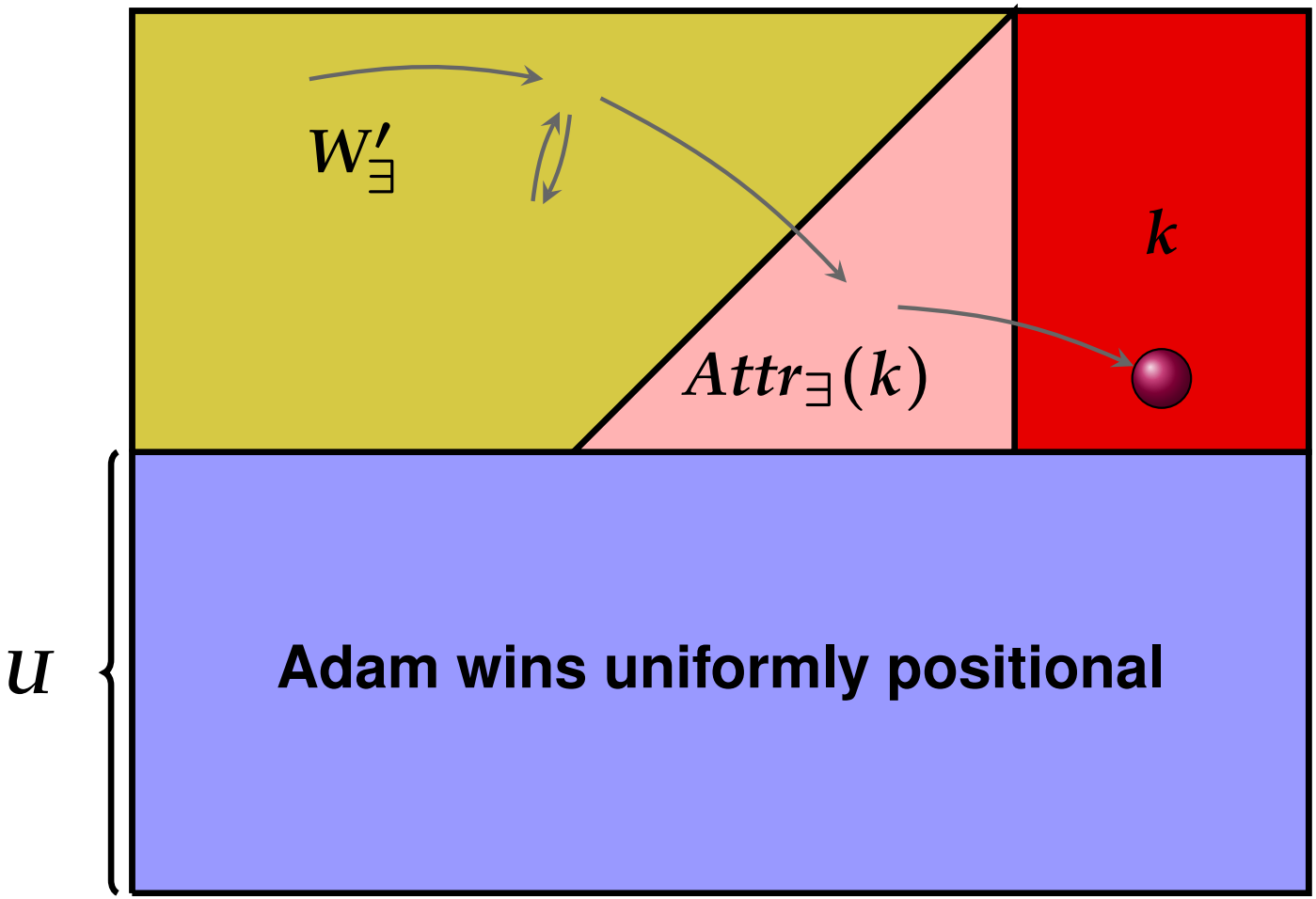
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

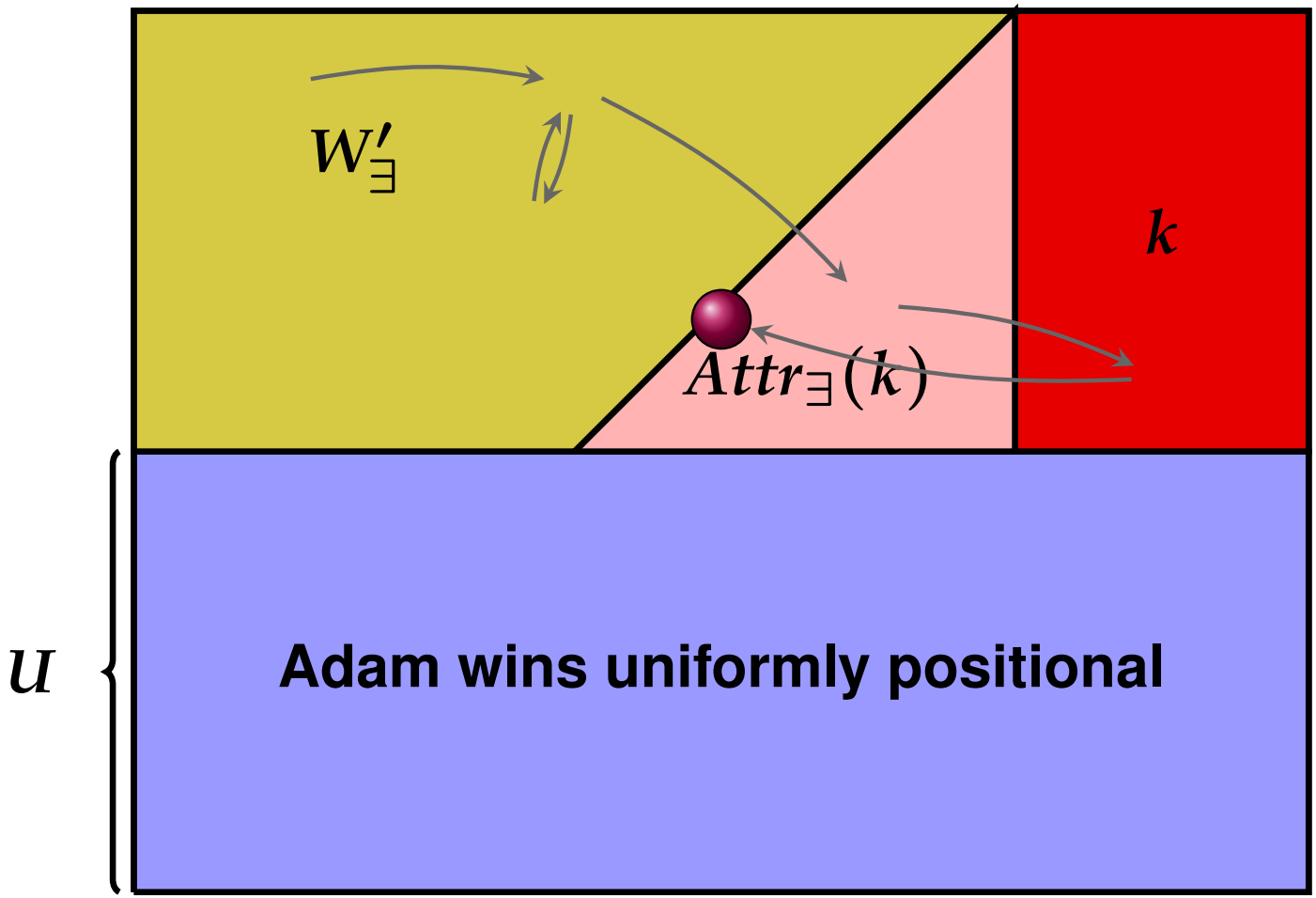
On the complement of U Eva has a uniform positional strategy.



Step 2 – Strategy for Eva

Lemma

On the complement of U Eva has a uniform positional strategy.



Remarks

- **The proof is non-constructive because we pick the set U of vertices from which Adam has a positional winning strategy.**
- **Now we see how to algorithmically solve parity games.**

Algorithms for Parity Games

An NP algorithm

Theorem

The decision problem "Given a parity game $\mathcal{G}$ and an initial vertex v_0, does Eva have a winning strategy from v_0?" is in NP $\cap$ co-NP.

An NP algorithm

Theorem

The decision problem "Given a parity game $\mathcal{G}$ and an initial vertex v_0, does Eva have a winning strategy from v_0?" is in NP $\cap$ co-NP.

Proof

Membership in NP:

1. **Guess a positional strategy for Eva**
2. **Verify that this strategy is winning from v_0**

Membership in co-NP follows from positional determinacy.

Verifying Strategies in Polynomial Time

Idea:

- Only consider edges relevant for the strategy.
- A positional strategy is losing for Eva if there is cycle reachable from v_0 on which the maximal priority k is odd.
- This means deleting all vertices of priority higher than k yields a graph containing a strongly connected component with highest priority k.
- Strongly connected components can be computed in polynomial time.
- We test the above criterion by successively deleting priorities.

Verifying Strategies – Algorithm

Let f be a positional strategy for Eva from v_0:

1. Remove all non-strategy edges of Eva
2. Restrict the resulting graph to vertices that are reachable from v_0
3. Remove vertices from the graph according to the following rule until the graph is empty:
   - 3.1 Compute the strongly connected components. Remove all the trivial components.
   - 3.2 Let k be the remaining maximal priority.
   - 3.3 If k is odd, then f is not winning. Stop.
   - 3.4 Otherwise, remove all vertices with priority k.

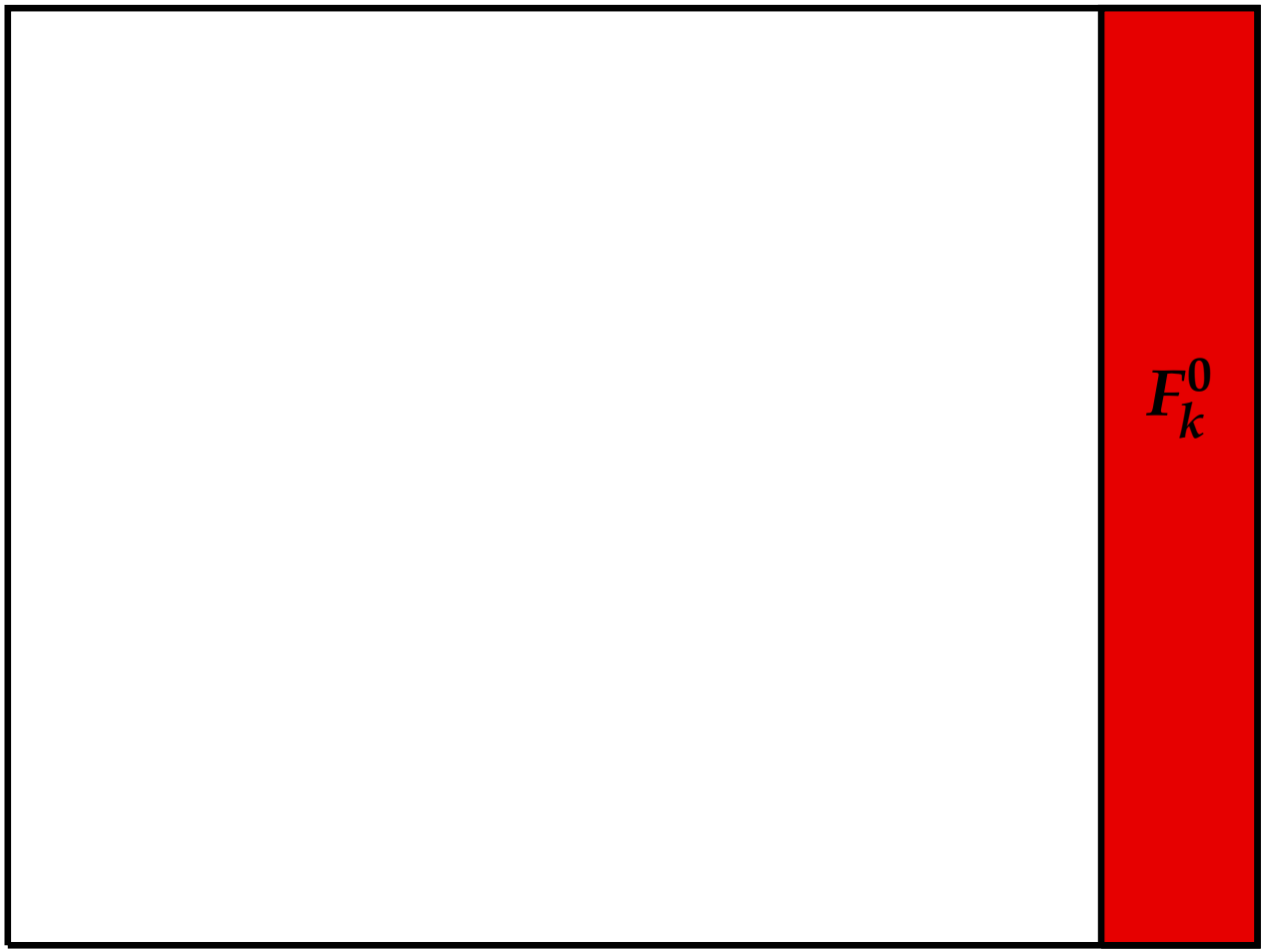
If the loop in 2. terminates with an empty graph, then f is winning.

Remarks on the Complexity

- **A deterministic algorithm based on the previous idea can test all positional strategies for Eva. The complexity is exponential in the size of the graph.**
- **By an inductive construction we can show that parity games can be solved in time that is only exponential in the number of priorities.**

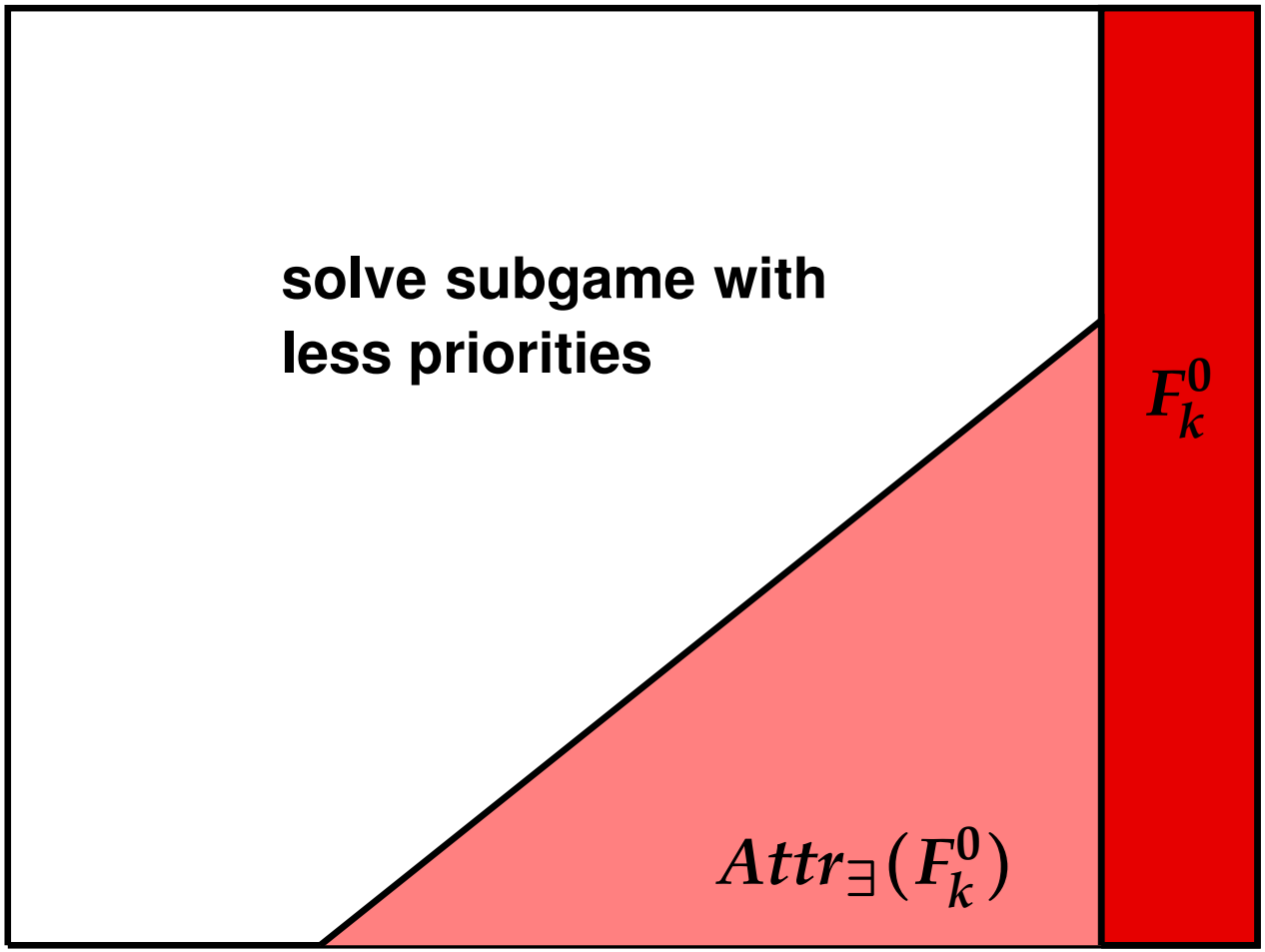
Inductive Construction – Idea

Highest priority k (assumed to be even)



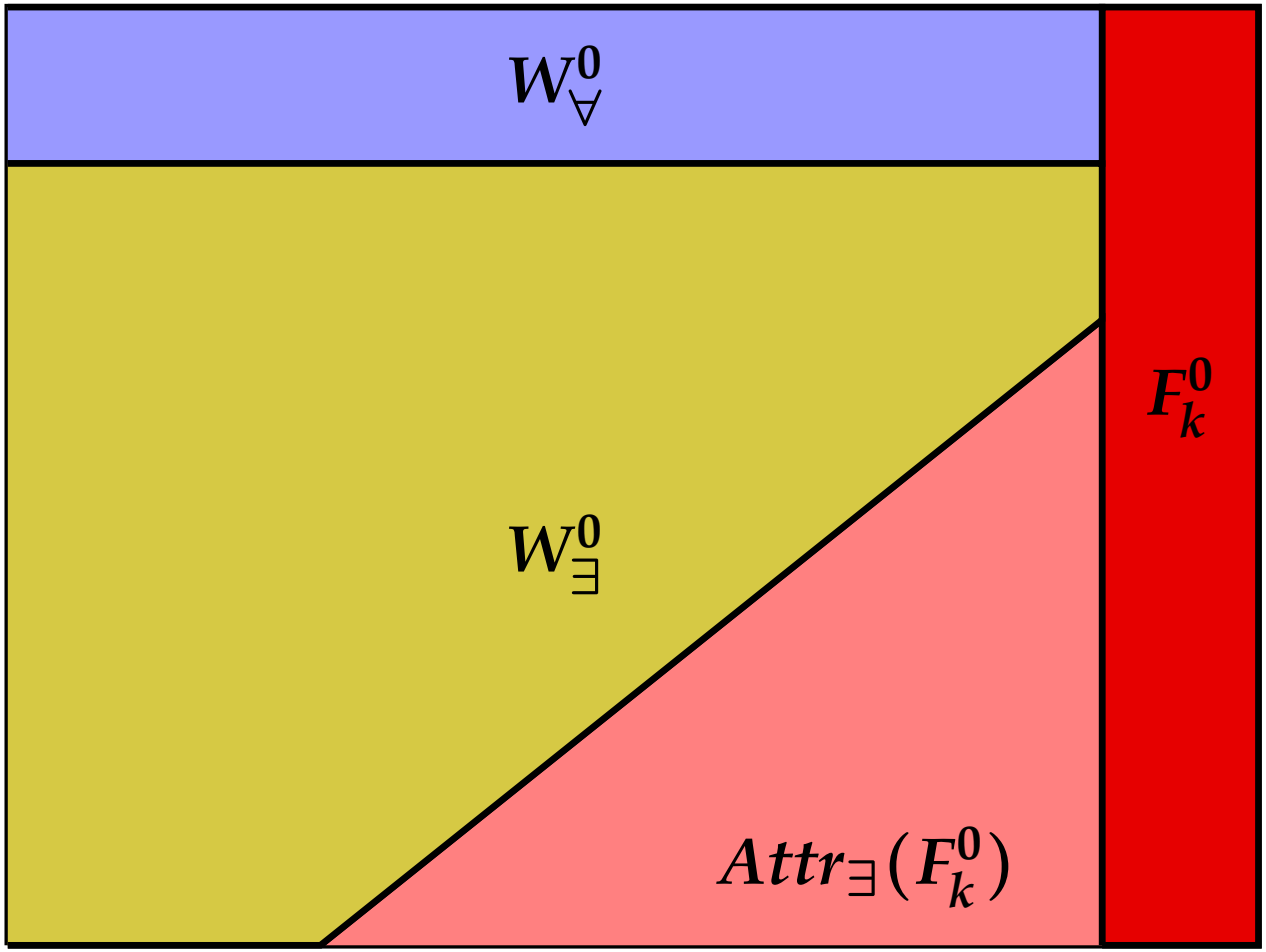
Inductive Construction – Idea

Highest priority k (assumed to be even)



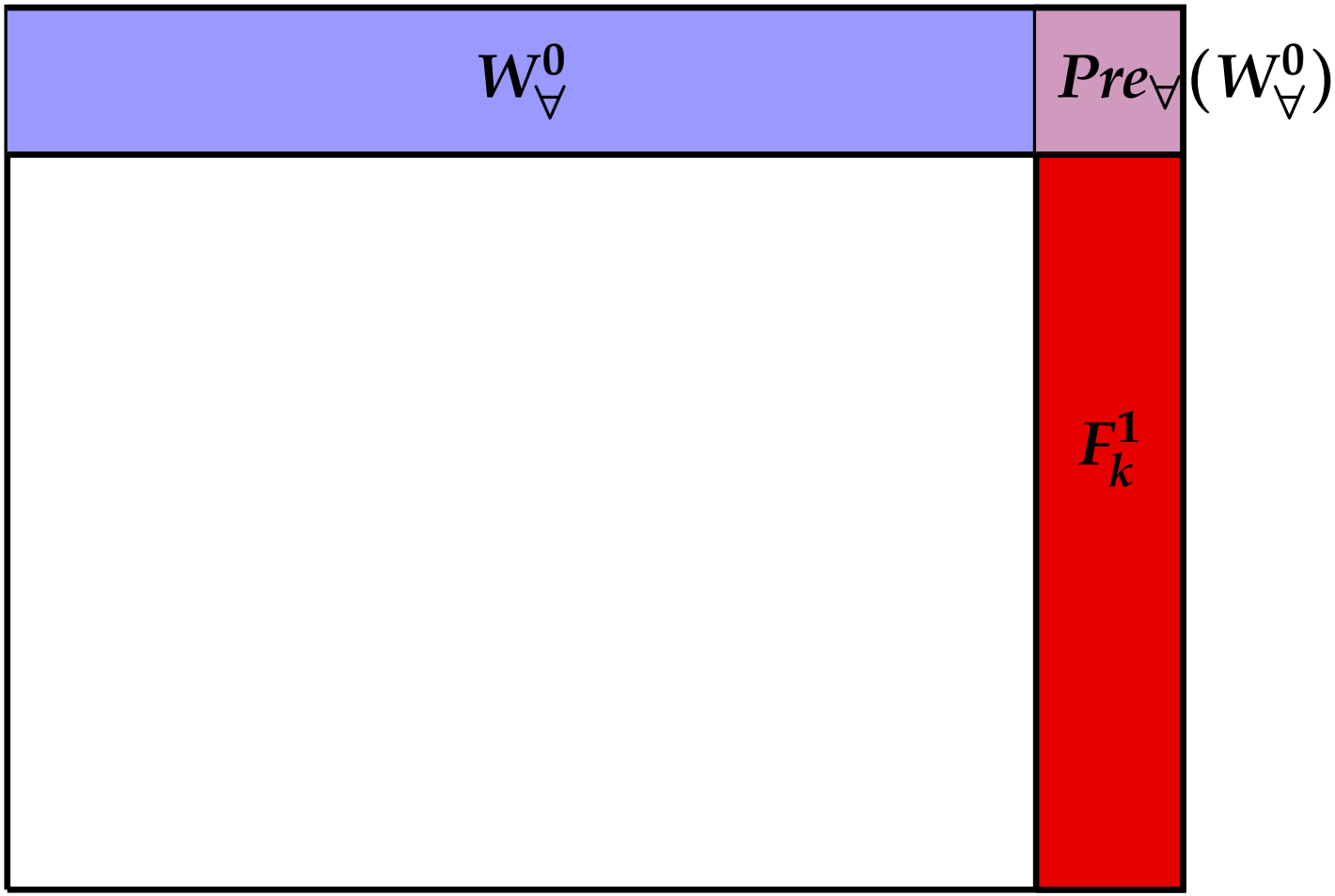
Inductive Construction – Idea

Highest priority k (assumed to be even)



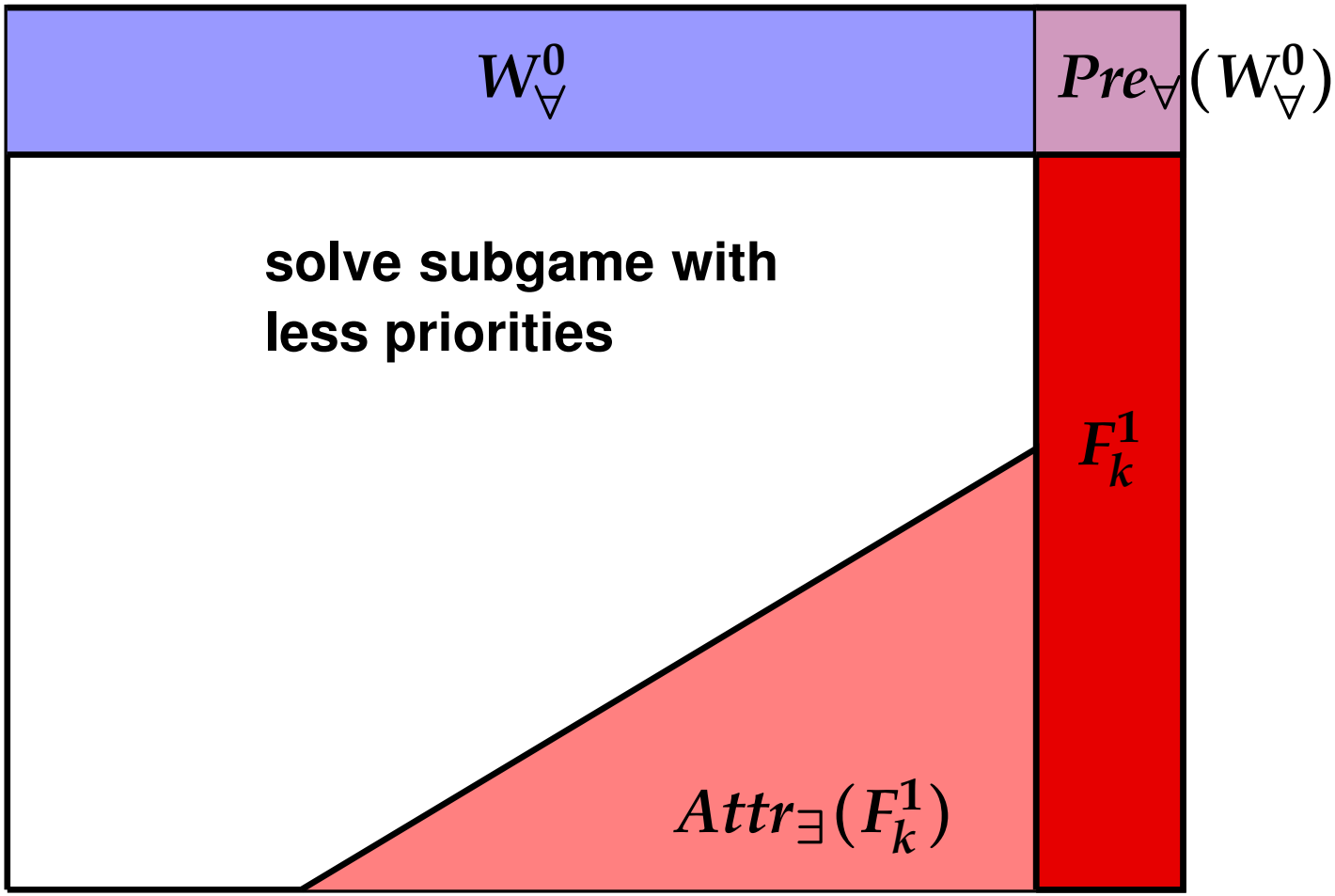
Inductive Construction – Idea

Highest priority k (assumed to be even)



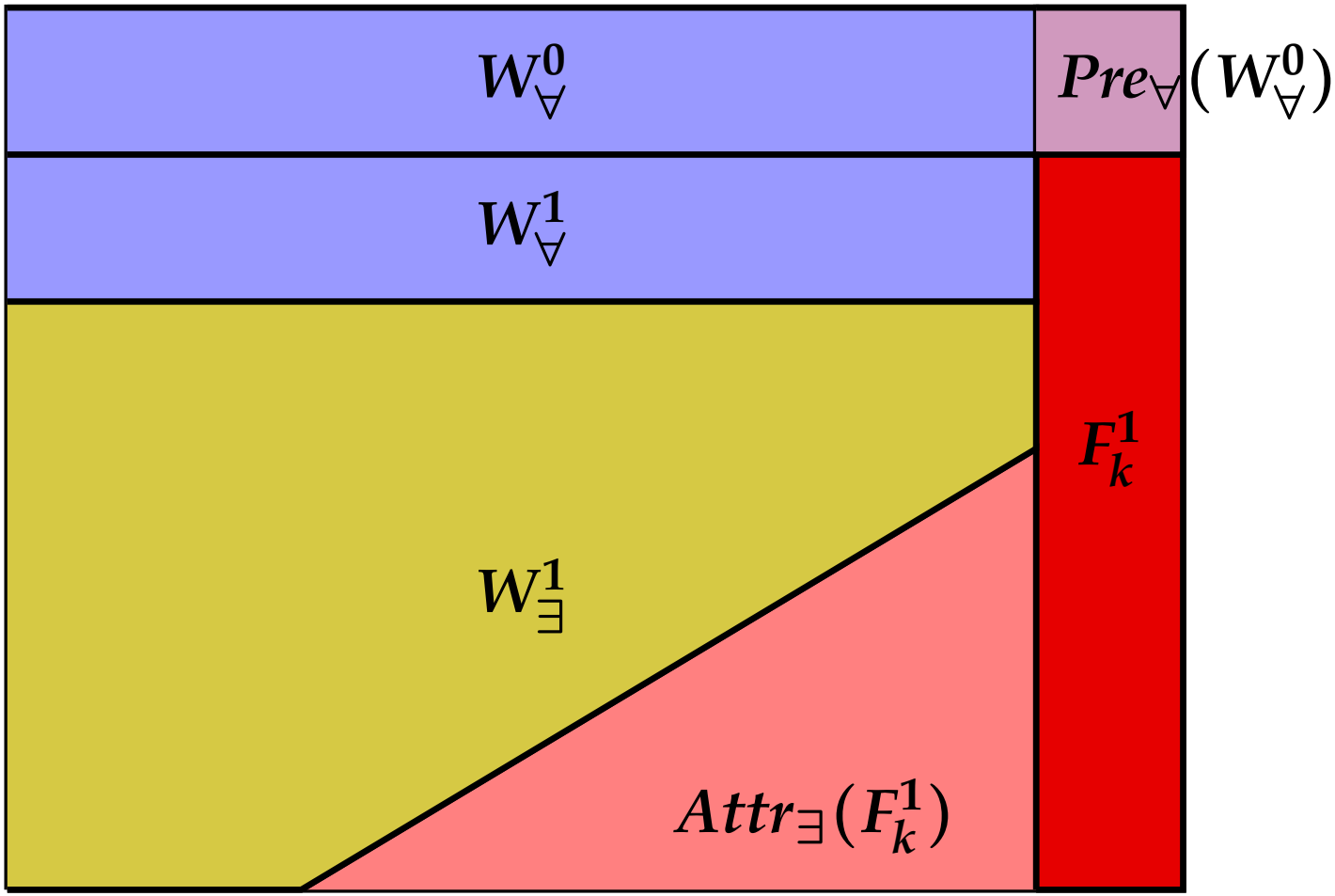
Inductive Construction – Idea

Highest priority k (assumed to be even)



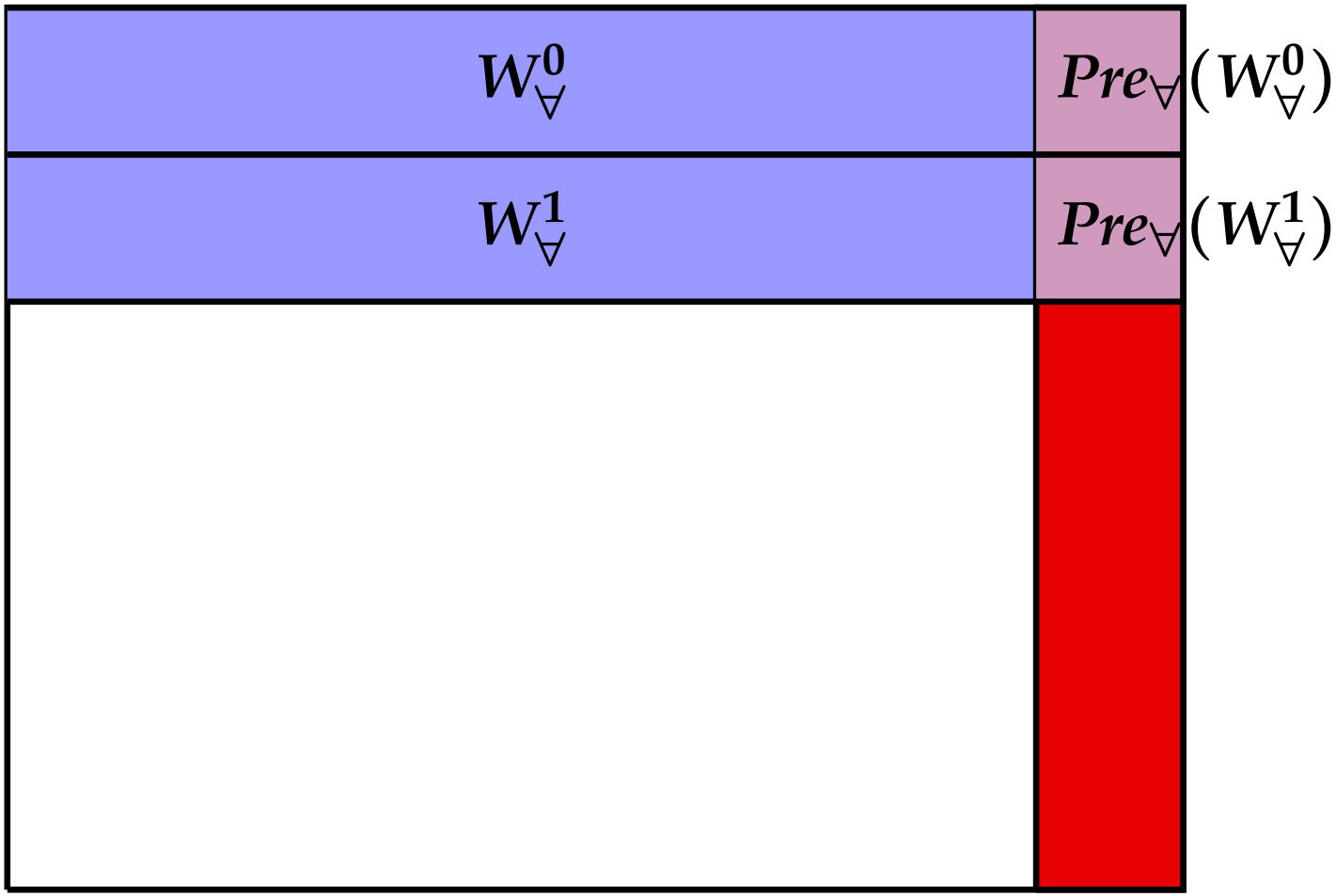
Inductive Construction – Idea

Highest priority k (assumed to be even)



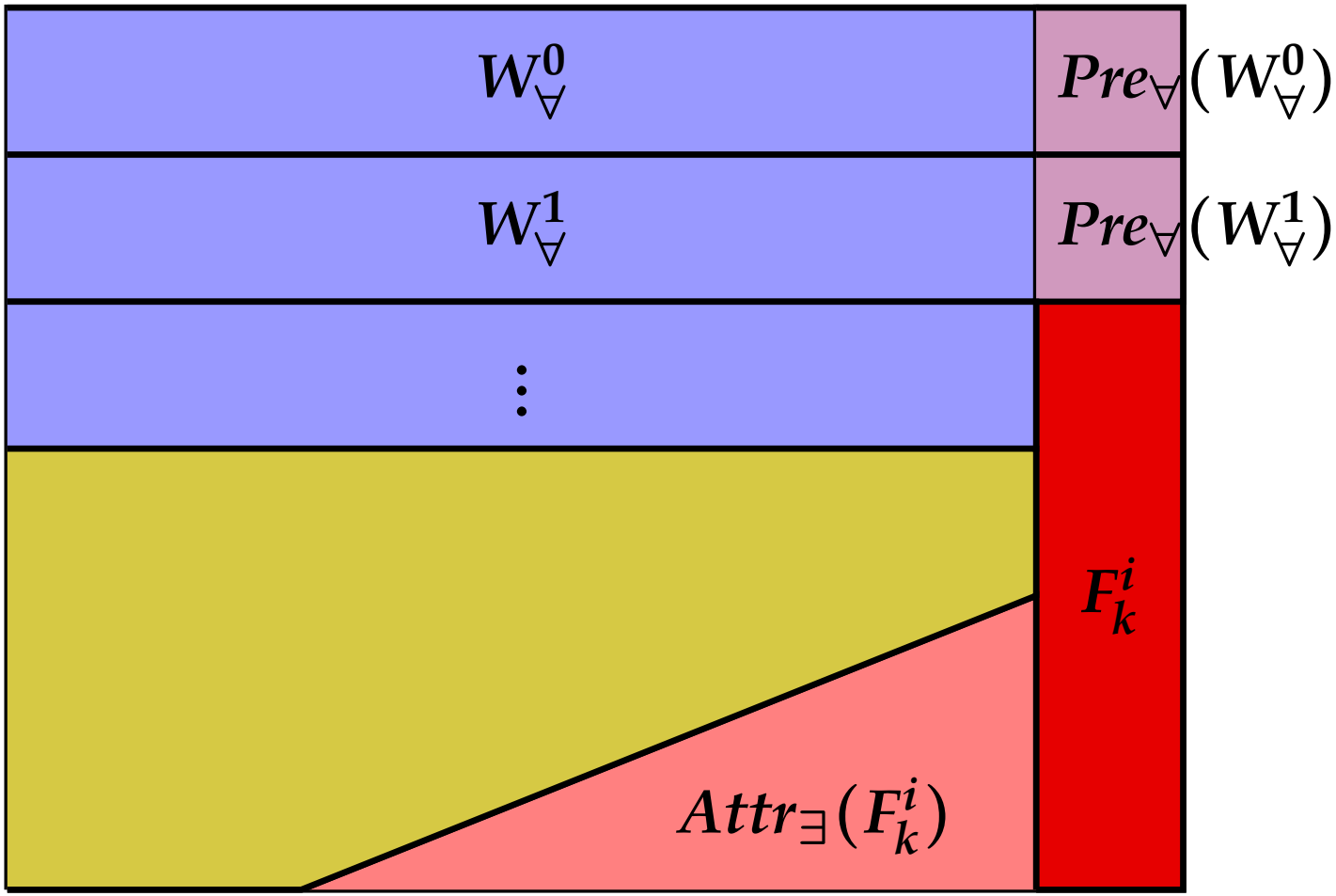
Inductive Construction – Idea

Highest priority k (assumed to be even)



Inductive Construction – Idea

Highest priority k (assumed to be even)



Complexity

Inductive construction for Graph of size n with k priorities:

- **Compute attractor: linear in n**
- **Solve subgame with $k-1$ priorities**
- **Number of iterations bounded by n**

$\rightsquigarrow \mathcal{O}(n^k)$

Other Algorithms

- **Small progress measures (Jurdziński 2000)**
  - $\mathcal{O}(n^{\frac{k}{2}})$
  - **easy to implement**
- **Strategy improvement (Vöge, Jurdziński 2000)**
  - **Runs fast in practice**
  - **No exponential lower bound known**

Other Algorithms

- Small progress measures (Jurdziński 2000)
  - $\mathcal{O}(n^{\frac{k}{2}})$
  - easy to implement
- Strategy improvement (Vöge, Jurdziński 2000)
  - Runs fast in practice
  - No exponential lower bound known

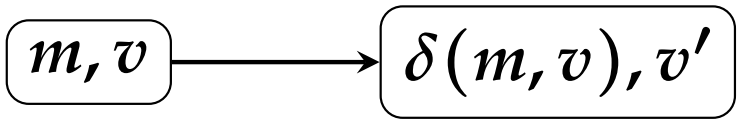
Open problem:

Can parity games be solved in polynomial time?

Reducing Muller Games to Parity Games

Recall – Schema of Game Reduction

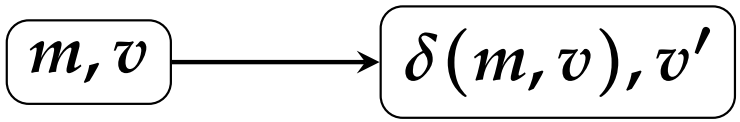
1. **Find a suitable memory structure $\mathcal{M}$.**
2. **Take the product of the game graph with the memory:**



3. **On the new graph define a new winning condition that is equivalent to the original winning condition. This new winning condition should be positionally determined.**
4. **Compute positional winning strategies on this new game.**
5. **Positional strategies on the extended graph correspond to strategies with memory $\mathcal{M}$ on the original graph.**

Recall – Schema of Game Reduction

1. Find a suitable memory structure $\mathcal{M}$.
2. Take the product of the game graph with the memory:



3. On the new graph define a new winning condition that is equivalent to the original winning condition. This new winning condition should be positionally determined.
4. Compute positional winning strategies on this new game.
5. Positional strategies on the extended graph correspond to strategies with memory $\mathcal{M}$ on the original graph.

Next goal: Find a suitable memory structure that allows to translate a Muller condition to a parity condition.

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

$$D \quad B \quad B \quad D \quad C \quad B \quad A \quad B \quad A \quad C \quad B \quad A \quad B \quad A \quad C$$

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A														
B														
C														
D														

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C

A	D
B	A
C	B
D	C

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C

A	D	B
B	A	D
C	B	A
D	C	C

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C

A	D	B	B
B	A	D	D
C	B	A	A
D	C	C	C

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C

A	D	B	B	D
B	A	D	D	B
C	B	A	A	A
D	C	C	C	C

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C

A	D	B	B	D	C
B	A	D	D	B	D
C	B	A	A	A	B
D	C	C	C	C	A

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B								
B	A	D	D	B	D	C								
C	B	A	A	A	B	D								
D	C	C	C	C	A	A								

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B	A							
B	A	D	D	B	D	C	B							
C	B	A	A	A	B	D	C							
D	C	C	C	C	A	A	D							

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B	A	B						
B	A	D	D	B	D	C	B	A						
C	B	A	A	A	B	D	C	C						
D	C	C	C	C	A	A	D	D						

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B	A	B	A					
B	A	D	D	B	D	C	B	A	B					
C	B	A	A	A	B	D	C	C	C					
D	C	C	C	C	A	A	D	D	D					

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B	A	B	A	C				
B	A	D	D	B	D	C	B	A	B	A				
C	B	A	A	A	B	D	C	C	C	B				
D	C	C	C	C	A	A	D	D	D	D				

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B	A	B	A	C	B			
B	A	D	D	B	D	C	B	A	B	A	C			
C	B	A	A	A	B	D	C	C	C	B	A			
D	C	C	C	C	A	A	D	D	D	D	D			

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B	A	B	A	C	B	A		
B	A	D	D	B	D	C	B	A	B	A	C	B		
C	B	A	A	A	B	D	C	C	C	B	A	C		
D	C	C	C	C	A	A	D	D	D	D	D	D		

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B	A	B	A	C	B	A	B	
B	A	D	D	B	D	C	B	A	B	A	C	B	A	
C	B	A	A	A	B	D	C	C	C	B	A	C	C	
D	C	C	C	C	A	A	D	D	D	D	D	D	D	

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B	A	B	A	C	B	A	B	A
B	A	D	D	B	D	C	B	A	B	A	C	B	A	B
C	B	A	A	A	B	D	C	C	C	B	A	C	C	C
D	C	C	C	C	A	A	D	D	D	D	D	D	D	D

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
B	A	D	D	B	D	C	B	A	B	A	C	B	A	B	A
C	B	A	A	A	B	D	C	C	C	B	A	C	C	C	B
D	C	C	C	C	A	A	D	D	D	D	D	D	D	D	D

Latest Appearance Record – Idea

- **Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
B	A	D	D	B	D	C	B	A	B	A	C	B	A	B	A
C	B	A	A	A	B	D	C	C	C	B	A	C	C	C	B
D	C	C	C	C	A	A	D	D	D	D	D	D	D	D	D

- **The colors appearing finitely often will gather at one end**
- **This allows to infer from the LAR which colors are visited infinitely often**

Latest Appearance Record

- A latest appearance record (LAR) over C is an ordering of the elements of C.

$$LAR(C) = \{[d_1 \cdots d_n] \mid d_i \in C \text{ and } d_i \neq d_j \text{ for all } i \neq j\}$$

- Memory update:

$$\delta([d_1 \cdots d_n], d) = [d d_1 \cdots d_{i-1} d_{i+1} \cdots d_n]$$

  for the unique i with $d = d_i$
- The LAR memory structure (for C) is defined as $\mathcal{M}_{LAR} = (LAR(C), \delta, m_0)$ for some arbitrary fixed LAR m_0.

Assigning Priorities

- **Assign priority depending on the size of the part of the LAR that changes in the next move**
- **The biggest part that changes infinitely often decides**
- **Example for $\mathcal{F} = \{\{B, D\}, \{A, B, C\}\}$**

D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
A	D	B	B	D	C	B	A	B	A	C	B	A	B	A	C
B	A	D	D	B	D	C	B	A	B	A	C	B	A	B	A
C	B	A	A	A	B	D	C	C	C	B	A	C	C	C	B
D	C	C	C	C	A	A	D	D	D	D	D	D	D	D	D
7	5	1	4	7	5	7	3	3	6	6	6	3	3	6	

From Muller to Parity

- **Let $\mathcal{G} = (G, \mathcal{F})$ be a Muller game with $|C| = n$ colors.**
- **Coloring function on game graph extended with LAR:**

$$c_{LAR}(v, [d_1 \cdots d_n]) = \begin{cases} 2i-1 & \{d_1, \ldots, d_i\} \notin \mathcal{F} \\ 2i & \{d_1, \ldots, d_i\} \in \mathcal{F} \end{cases}$$

  with $c(v) = d_i$
- **This defines a parity game equivalent to the Muller game as required for a game reduction.**

Determinacy of Muller Games

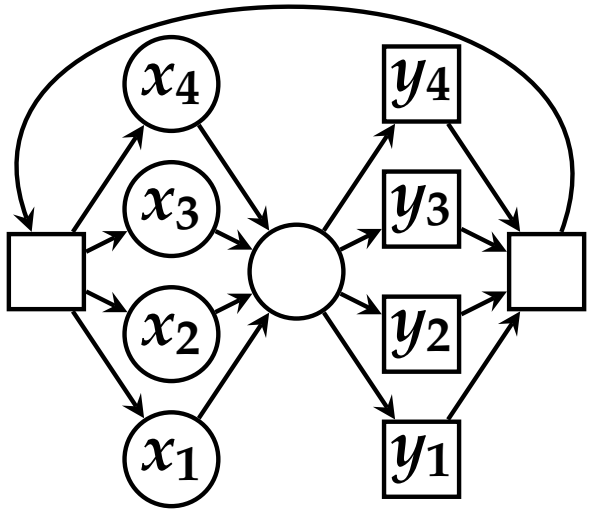
Because parity games are positionally determined with uniform strategies, we obtain the following.

Theorem

Muller games are determined and both players have uniform strategies with memory of size at most $|C|!$ on their winning areas, where $|C|$ is the number of colors of the Muller game.

Initial Example

$$\alpha \in \mathit{Win} \text{ iff } \max\{i \mid y_i \in \mathrm{Inf}(\alpha)\} = |\{i \mid x_i \in \mathrm{Inf}(\alpha)\}|$$



Simple strategy based on LAR idea:

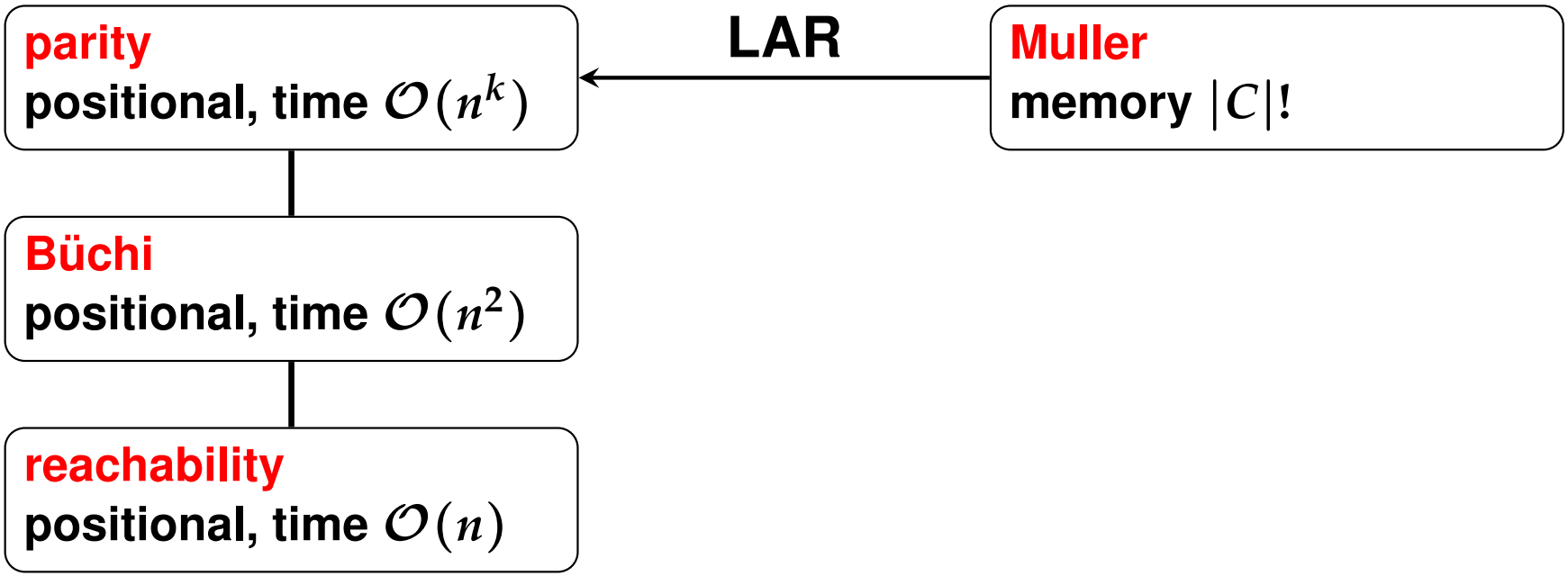
- **Eva maintains an LAR of the x_i together with the size j of the portion that changed in the last update**
- **She chooses the y_j corresponding to this size**

Summary

- **Parity conditions**
- **Uniform positional determinacy**
- **Membership in NP $\cap$ co-NP**
- **Inductive construction (time $\mathcal{O}(n^k)$)**
- **Reduction from Muller to parity using LAR**

Summary

- Parity conditions
- Uniform positional determinacy
- Membership in NP ∩ co-NP
- Inductive construction (time $\mathcal{O}(n^k)$)
- Reduction from Muller to parity using LAR



1 Basics

2 Positional Strategies: Reachability and Attractors

3 Büchi Games

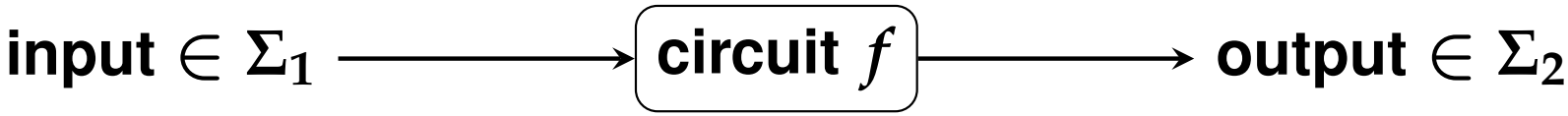
4 Muller Games and Finite Memory Strategies

5 Parity Games

- Positional Determinacy
- Algorithms for Parity Games
- Reducing Muller Games to Parity Games

6 Applications

Synthesis



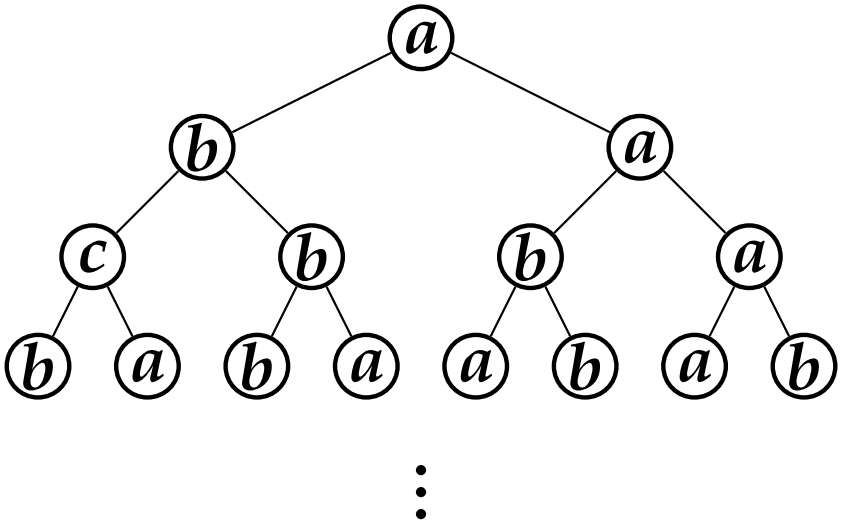
$$\varphi(\alpha, \beta)$$

Synthesis

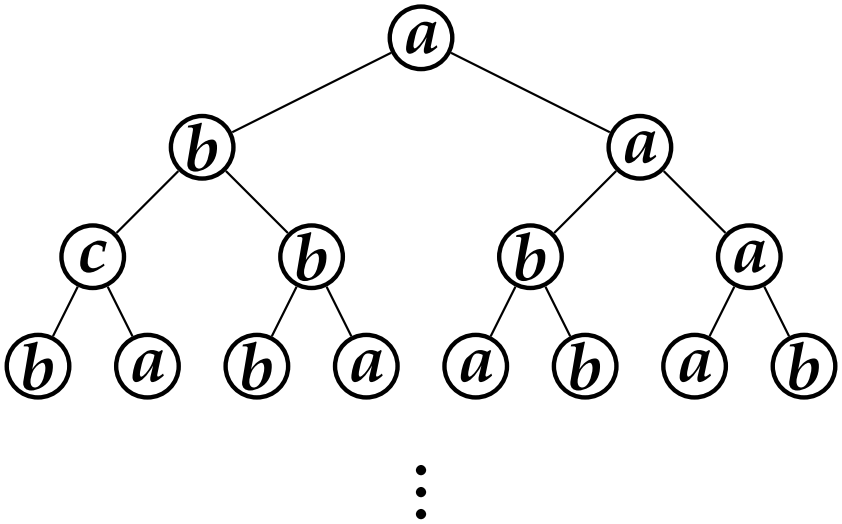
$$\varphi(\alpha, \beta)$$

- Model the problem by a game graph in which Adam chooses input symbols from Σ_1 and Eva chooses output symbols from Σ_2
- Transform specification into an equivalent deterministic Muller or parity automaton (not treated in this tutorial)
- Take the product of game graph and automaton (as in the schema of game reduction)
- Solve the game with the presented methods
- The resulting strategy with memory is a realization for φ

Automata on Infinite Trees



Automata on Infinite Trees

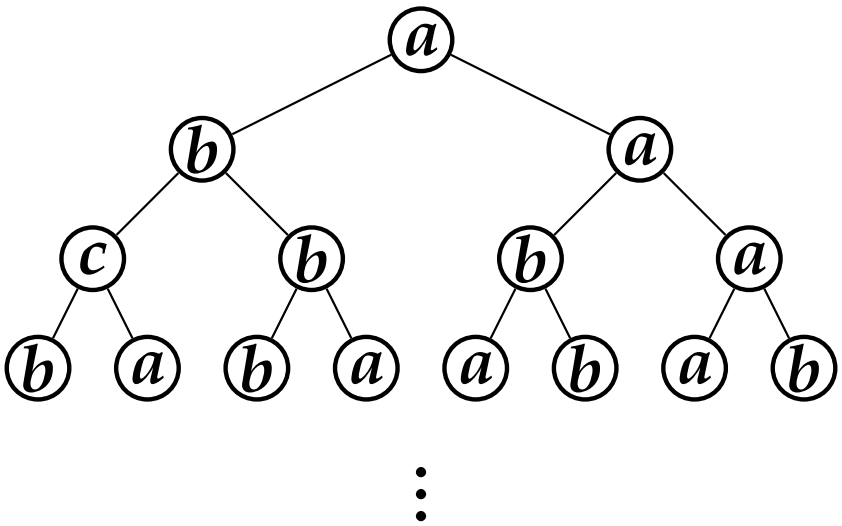


transitions of the form:

$$q, a \quad / \ \backslash \quad q_1 \ \ q_2$$

with states q, q_1, q_2

Automata on Infinite Trees



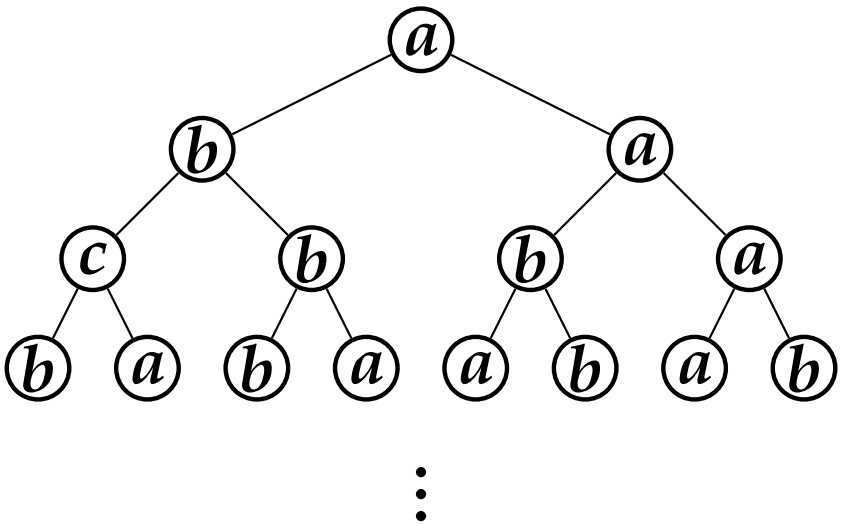
transitions of the form:

$$q, a \quad / \ \backslash \quad q_1 \ \ q_2$$

with states q, q_1, q_2

- **Run: labelling of tree with states, build from transitions**
- **Acceptance condition: Muller / parity on each infinite branch of the tree**
- **Automaton accepts if there is an accepting run**

Automata on Infinite Trees



transitions of the form:

$$q, a \quad / \ \backslash \quad q_1 \ \ q_2$$

with states q, q_1, q_2

- **Run: labelling of tree with states, build from transitions**
- **Acceptance condition: Muller / parity on each infinite branch of the tree**
- **Automaton accepts if there is an accepting run**

These automata are a powerful tool when dealing with logics on branching structures (they are as expressive as monadic second-order logic over the infinite binary tree).

Automata and Games

- **Acceptance of a tree t can be characterized by a game. The arena is infinite because t is infinite:**
  - **The game starts at the root in the initial state of the automaton. Then the following is repeated to infinity:**
  - **Eva plays a transition of the automaton**
  - **Adam chooses a direction in the tree and the corresponding state of the transition**

Automata and Games

- Acceptance of a tree t can be characterized by a game. The arena is infinite because t is infinite:
  - The game starts at the root in the initial state of the automaton. Then the following is repeated to infinity:
  - Eva plays a transition of the automaton
  - Adam chooses a direction in the tree and the corresponding state of the transition
- Eva wins if the resulting infinite sequence of states satisfies the acceptance condition of the automaton.
- Eva has winning strategy iff there is an accepting run on t
- This is used for complementing these automata: the automaton for the complement checks if Adam has a winning strategy.

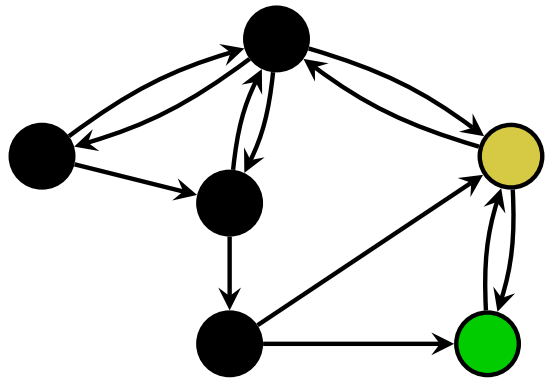
Model Checking

Checking whether a formula of a temporal or modal logic (like the modal μ-calculus) holds on a finite transition system can be reduced to games.

Model Checking

Checking whether a formula of a temporal or modal logic (like the modal μ-calculus) holds on a finite transition system can be reduced to games.

$$\mu X(\bullet \vee \Box X \vee (\bullet \wedge \Diamond X))$$



Construct model checking game as a product of formula and transition system.

Parity Games and μ-Calculus

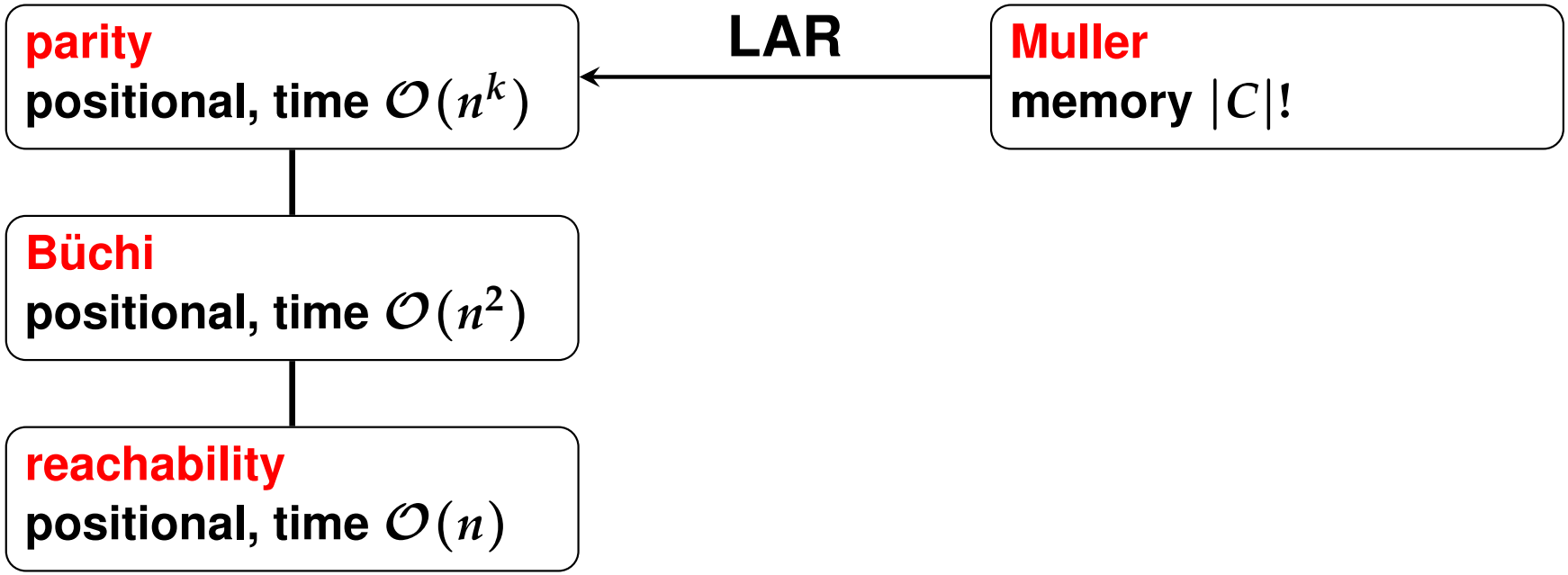
Theorem

Model checking for modal μ-calculus is polynomial time equivalent to the problem of solving parity games.

Finish

Conclusion

- **Basic theory of two player games of infinite duration**



- **Applications in synthesis, tree automata, model checking**
- **Many possible extensions: Probabilities, concurrency, time, algorithms for classes of infinite graphs,...**

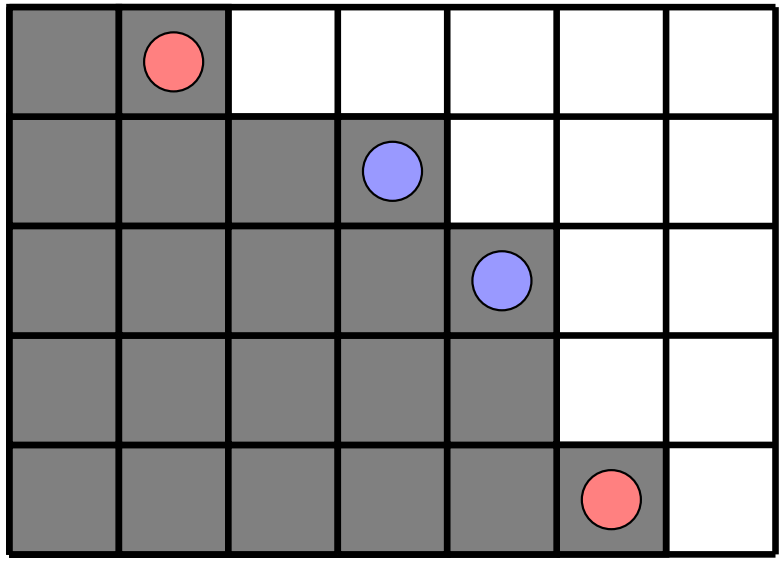
References

There are good collections/surveys:

- ***Automata, Logics, and Infinite Games.*** **LNCS 2500, 2002**
- **W. Zielonka.** ***Infinite games on finitely coloured graphs with applications to automata on infinite trees.*** **TCS 200, 1998.**
- **W. Thomas.** ***Languages, Automata, and Logic.*** **Handbook of Formal Language Theory. 1997**

Reminder

Don’t forget to think about this:



Challenge: Show that one of the players (the one who starts or the other) has a winning strategy for every size of the board.

The proof is very short and uses positional determinacy of reachability games. But it does not provide the strategy...