Infinite games and automata theory

Christof Löding

RWTH Aachen, Germany

Spring School 2009

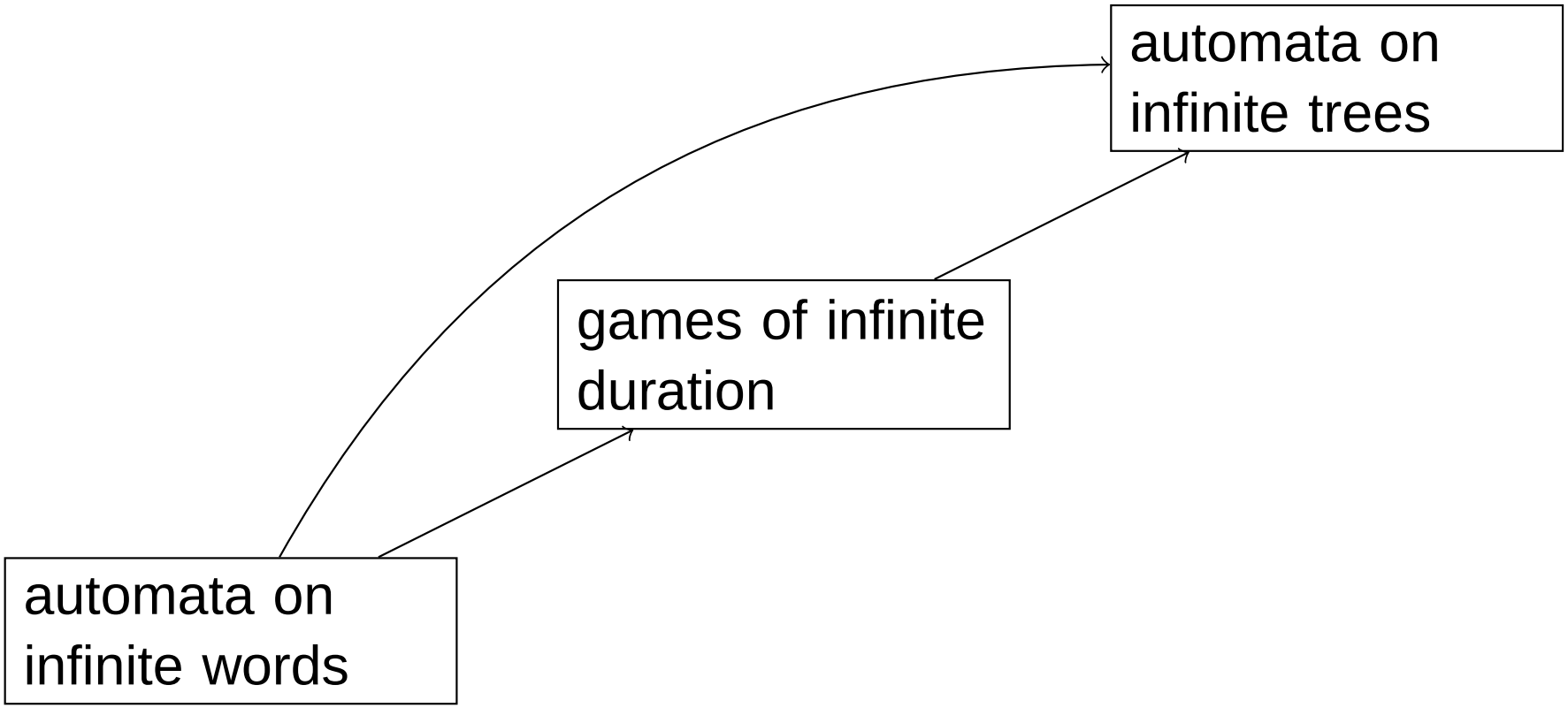
June 1–5, Bertinoro, Italy

In this tutorial

1. How can automata theory help to solve problems for games?
2. How can games help to solve problems in automata theory?

In this tutorial

1. How can automata theory help to solve problems for games?
2. How can games help to solve problems in automata theory?



1 Introduction

2 Basics on games

3 Transformation of winning conditions

- ω-Automata
- Game reductions
- Logical winning conditions

4 Tree automata

- Complementation
- Emptiness

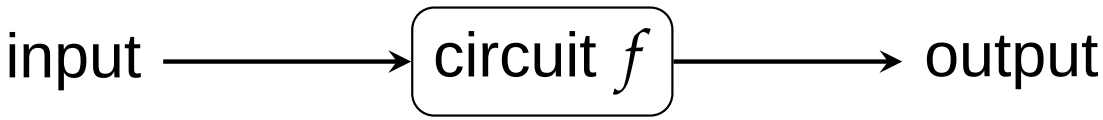
5 Beyond finite automata

Origin

Circuit synthesis and Church's problem (1957)

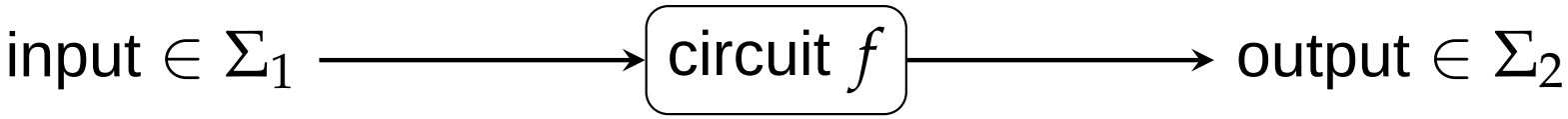
Setting:

- 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

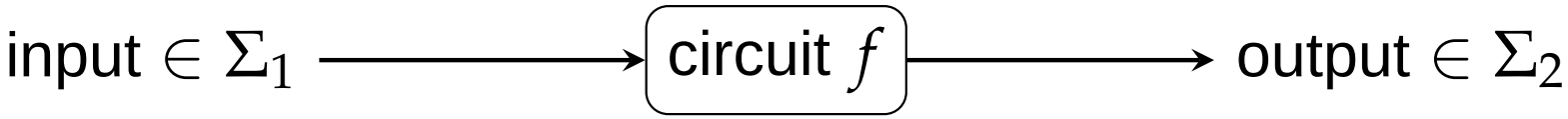


- Input sequence $\alpha \in \Sigma_1^\omega$ and output sequence $\beta \in \Sigma_2^\omega$
- Specification $\varphi(\alpha, \beta)$

Problem:

- Decide if the there is a sequential transformation $f : \Sigma_1^* \to \Sigma_2$ 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 $f : \Sigma_1^* \to \Sigma_2$ 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 presents the automata theoretic aspects

1 Introduction

2 Basics on games

3 Transformation of winning conditions

- ω-Automata
- Game reductions
- Logical winning conditions

4 Tree automata

- Complementation
- Emptiness

5 Beyond finite automata

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

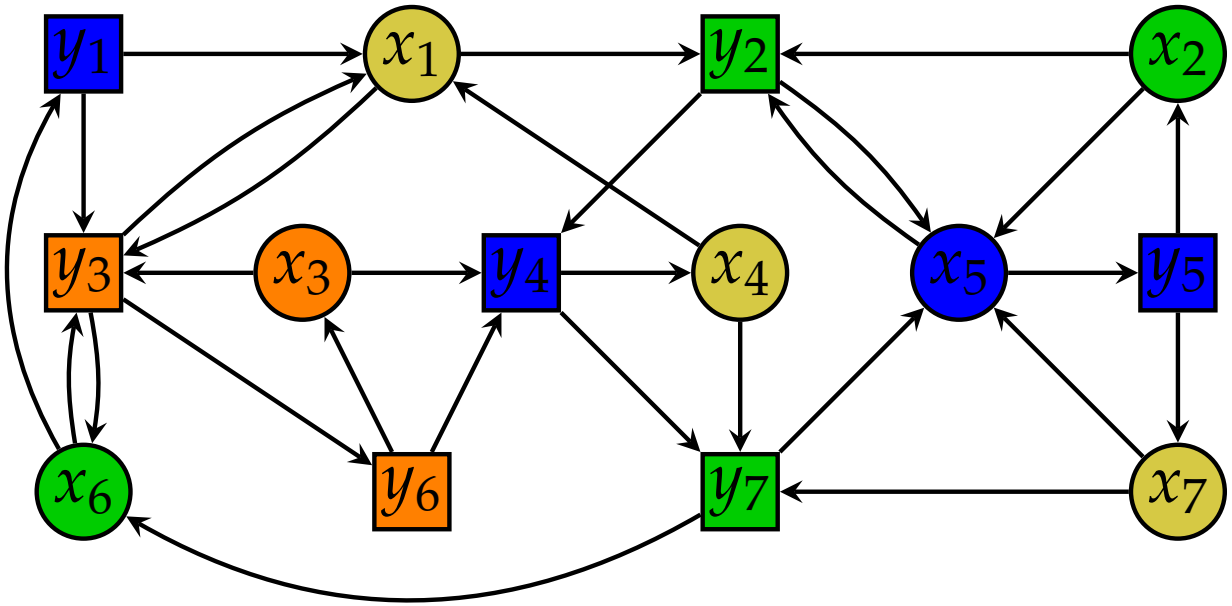
$G = (V_{\mathsf{E}}, V_{\mathsf{A}}, E, c)$

- V_{E}: vertices of Eva (player 1, circle)
- V_{A}: vertices of Adam (player 2, box)
- $E \subseteq V \times V$: edges with $V = V_{\mathsf{E}} \cup V_{\mathsf{A}}$
- $c : V \to C$ with a finite set of colors C

Game Graph / Arena

$G = (V_{\mathsf{E}}, V_{\mathsf{A}}, E, c)$

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



Plays

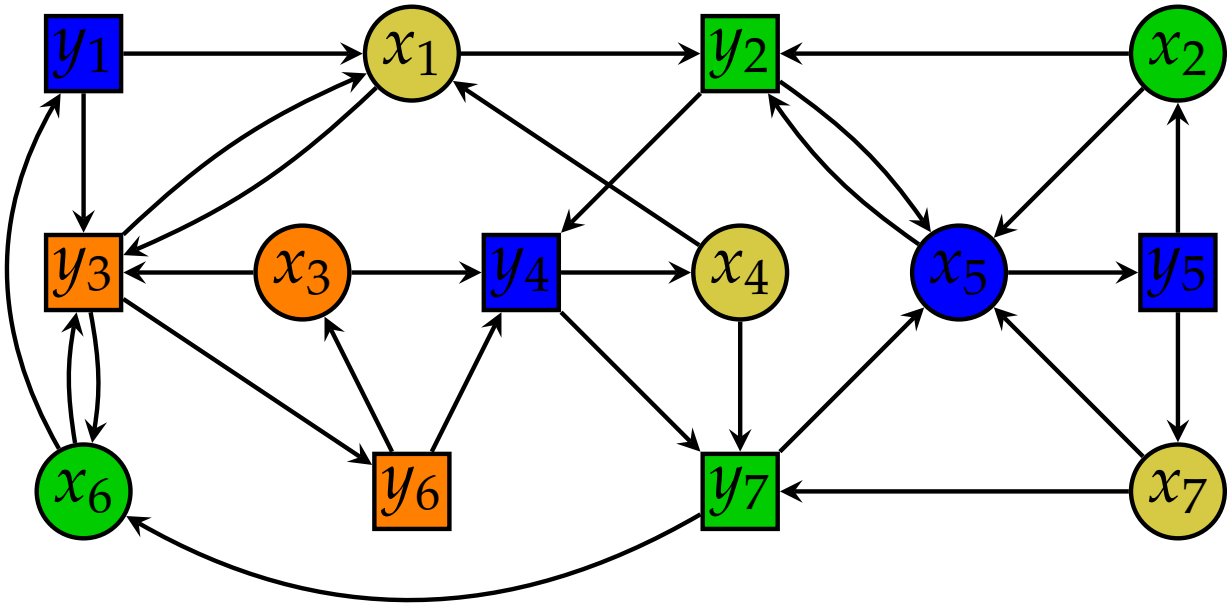
A **play** in G is an infinite sequence $\alpha = v_0 v_1 v_2 \cdots$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$.

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 $i \geq 0$.

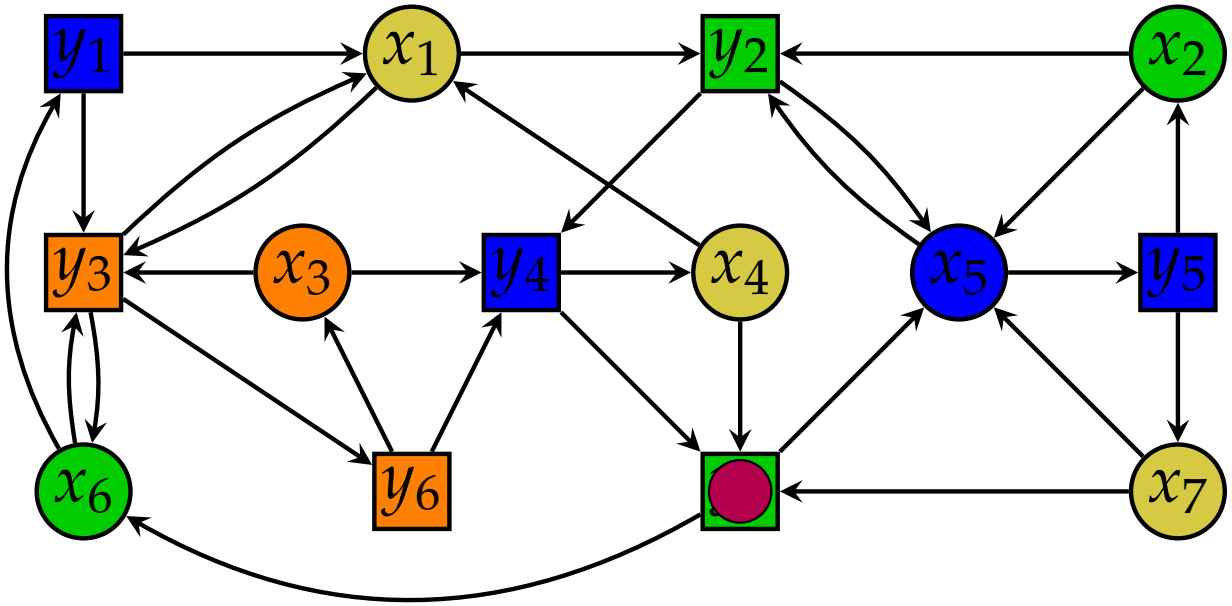
By $c(\alpha)$ we denote the corresponding sequence of colors $c(v_0)c(v_1)c(v_2)\cdots$



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 $c(v_0)c(v_1)c(v_2)\cdots$



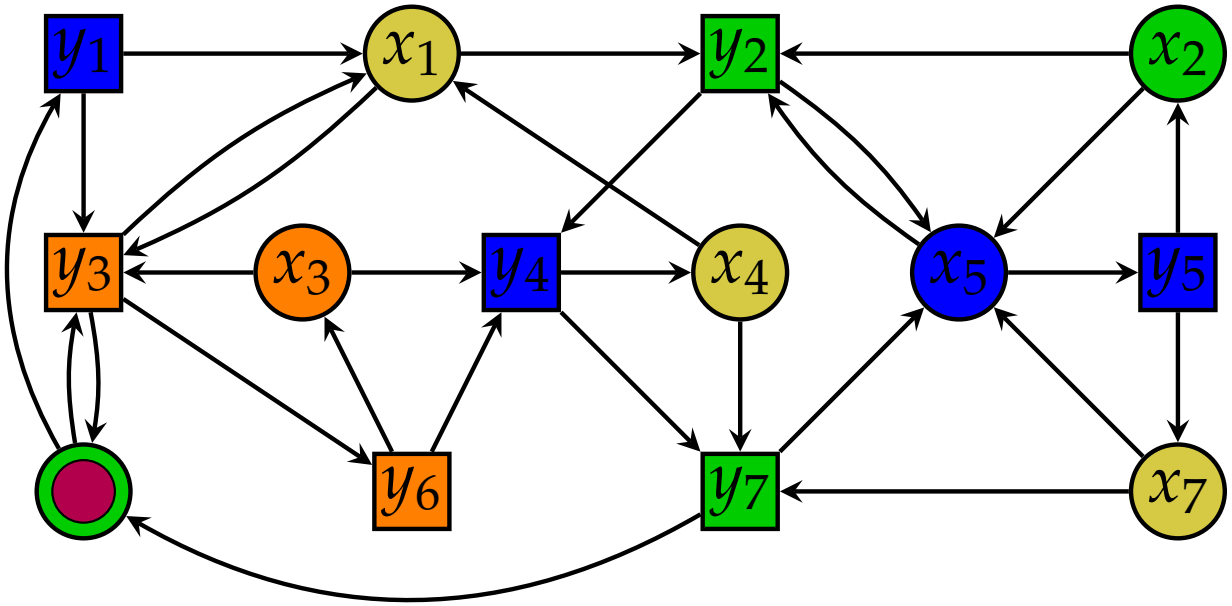
α : y_7

$c(\alpha)$: ●

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 $c(v_0)c(v_1)c(v_2)\cdots$



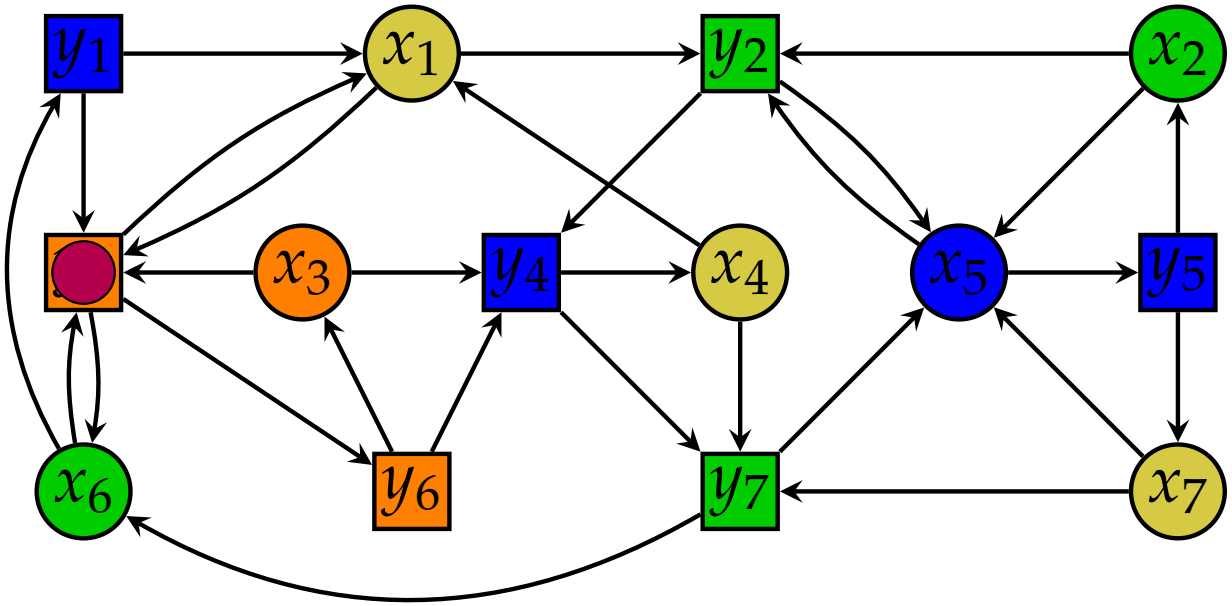
α : y_7 x_6

$c(\alpha)$: ● ●

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 $c(v_0)c(v_1)c(v_2)\cdots$



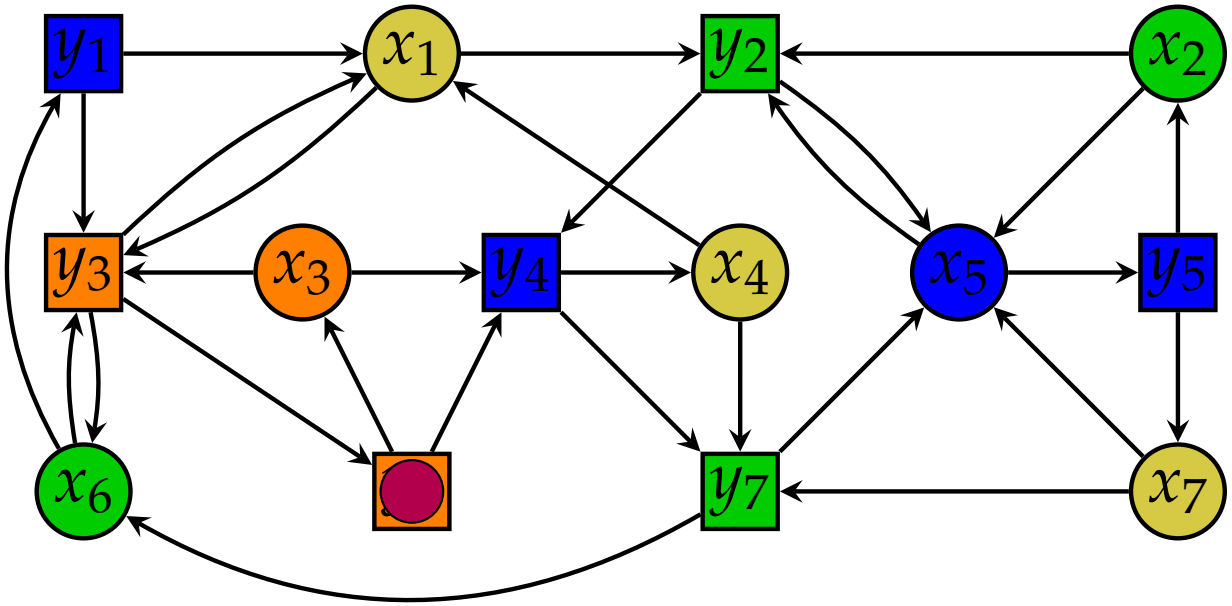
α : y_7 x_6 y_3

$c(\alpha)$: ● ● ●

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 $c(v_0)c(v_1)c(v_2)\cdots$



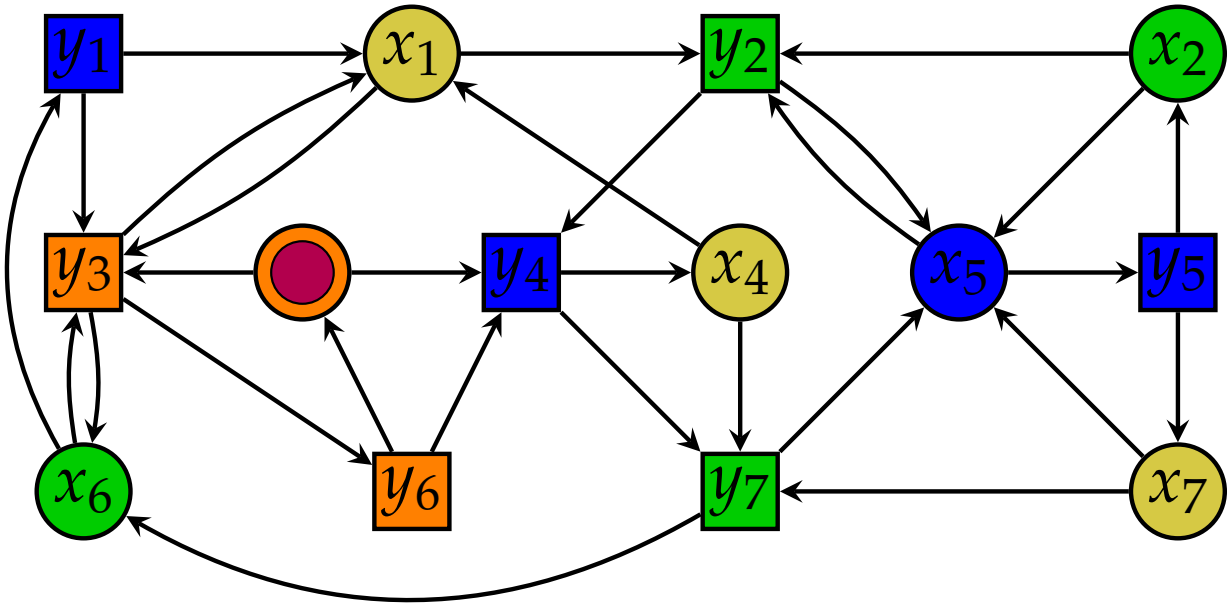
α : y_7 x_6 y_3 y_6

$c(\alpha)$: ● ● ● ●

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 $c(v_0)c(v_1)c(v_2)\cdots$



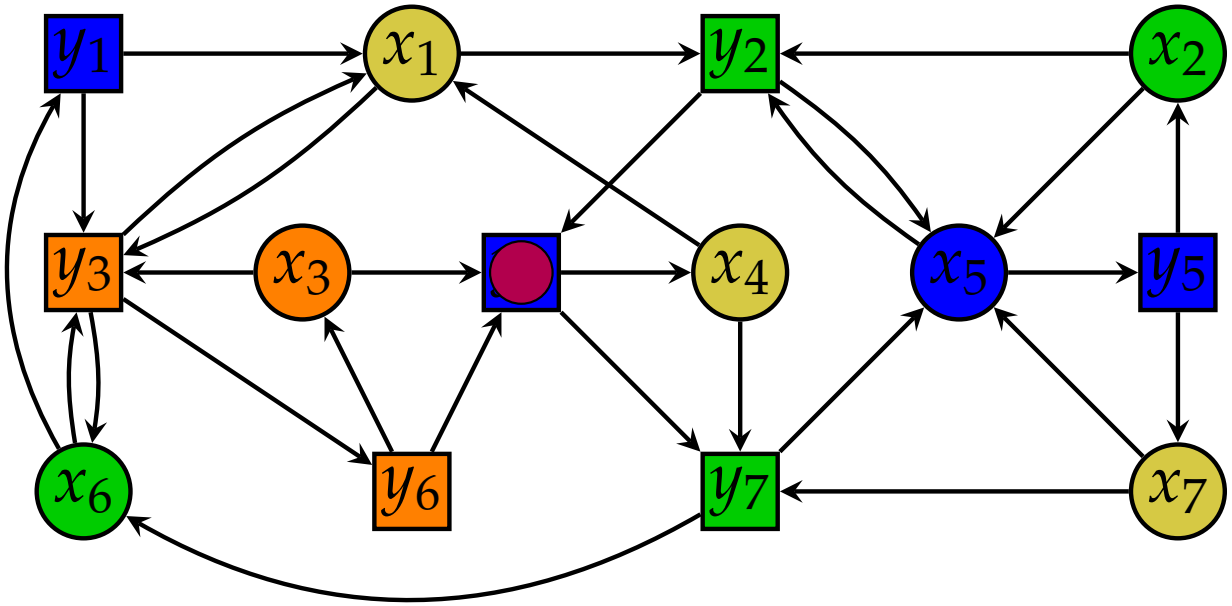
α : y_7 x_6 y_3 y_6 x_3

$c(\alpha)$: ● ● ● ● ●

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

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

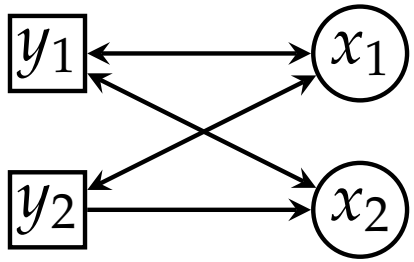
- G: game graph
- $Win \subseteq C^\omega$: winning condition

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

Examples for winning conditions:

- **Büchi**: given by $F \subseteq C$
  Win contains all plays α with $\mathrm{Inf}(\alpha) \cap F \neq \emptyset$
- **Muller**: given by $\mathcal{F} \subseteq 2^C$
  Win contains all plays α with $\mathrm{Inf}(\alpha) \in \mathcal{F}$
- **Parity**: set of colors is a finite set of natural numbers (priorities)
  Win contains all plays α with $\max(\mathrm{Inf}(c(\alpha)))$ even

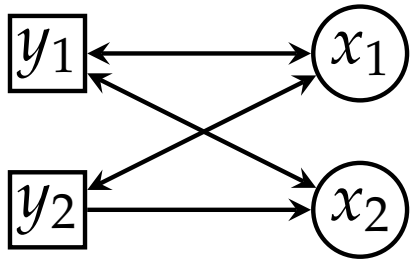
Trying to Win – Strategies



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



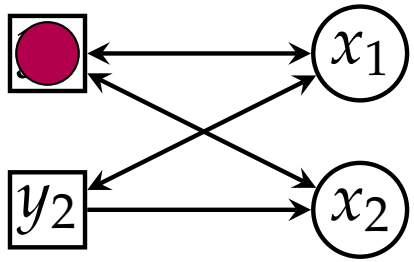
Winning condition for Eva: y_2 is visited infinitely often iff x_1 and x_2 are both visited infinitely often.

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

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.

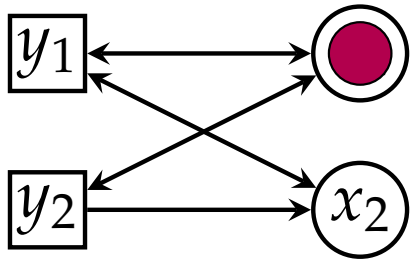
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

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.

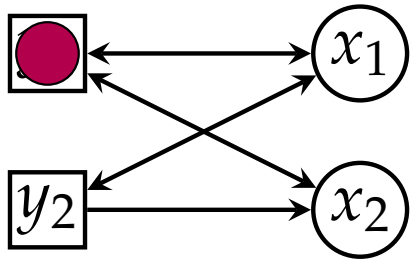
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

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.

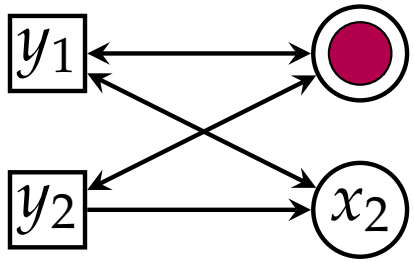
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

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.

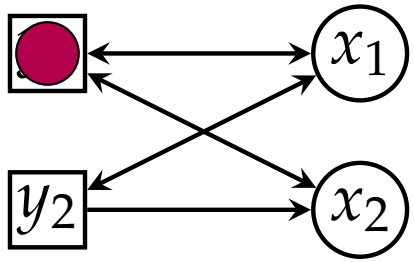
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

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.

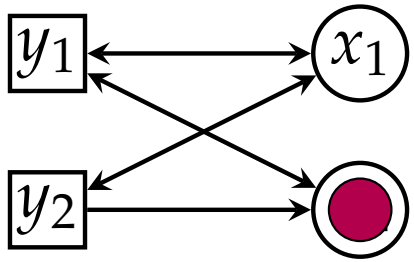
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

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.

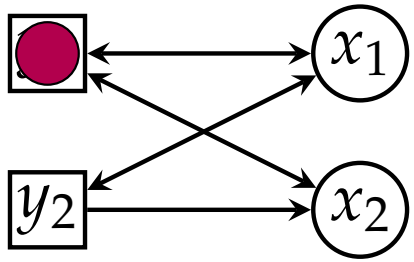
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

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.

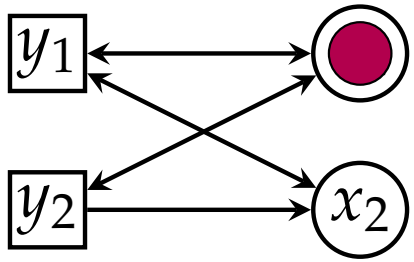
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

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.

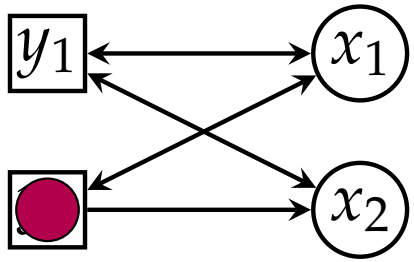
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

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.

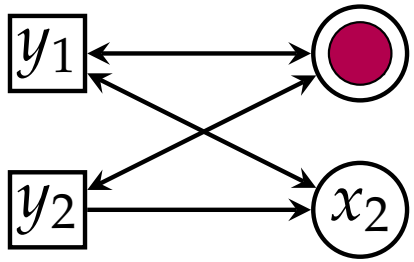
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

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.

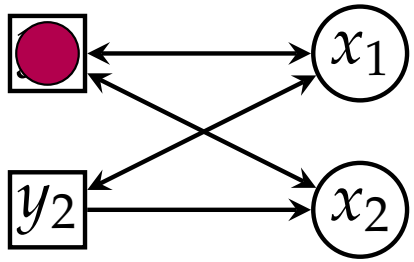
$$y_2 \in \mathrm{Inf}(\alpha) \Leftrightarrow \{x_1, x_2\} \subseteq \mathrm{Inf}(\alpha)$$

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

$$\sigma : V^* V_{\mathsf{E}} \to V$$

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

Strategies – Formal

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

$$\sigma : V^* V_{\mathsf{E}} \to V$$

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

$$\forall i : v_i \in V_{\mathsf{E}} \to \sigma(v_0 \cdots v_i) = v_{i+1}$$

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

Strategies – Formal

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

$$\sigma : V^* V_{\mathsf{E}} \to V$$

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

$$\forall i : v_i \in V_{\mathsf{E}} \to \sigma(v_0 \cdots v_i) = v_{i+1}$$

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

Determinacy

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

Determinacy

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

Determinacy is about swapping quantifiers

Eva does not have a winning strategy:

$$\forall \sigma_{\mathsf{E}} \exists \sigma_{\mathsf{A}} \ (\ \sigma_{\mathsf{A}} \text{ wins against } \sigma_{\mathsf{E}})$$

Determinacy

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

Determinacy is about swapping quantifiers

Eva does not have a winning strategy:

$$\forall \sigma_{\mathsf{E}} \exists \sigma_{\mathsf{A}} \ (\ \sigma_{\mathsf{A}} \text{ wins against } \sigma_{\mathsf{E}})$$

By determinacy:

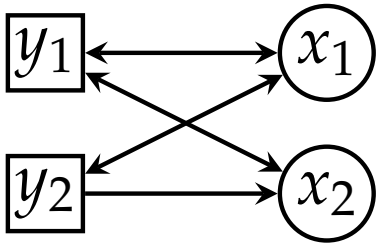
$$\exists \sigma_{\mathsf{A}} \forall \sigma_{\mathsf{E}} \ (\ \sigma_{\mathsf{A}} \text{ wins against } \sigma_{\mathsf{E}})$$

Special strategies

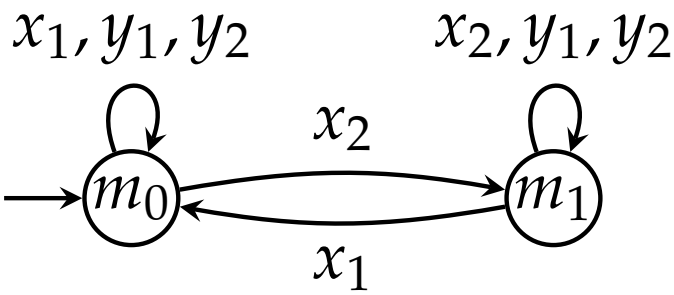
- **Computable**: The function $\sigma : V^* V_{\mathsf{E}} \to V$ is computable.
- **Positional**: The strategy only depends on the current vertex (not on the past), i.e., $\sigma : V_{\mathsf{E}} \to V$.
- **Finite memory**: The strategy is implemented by a deterministic finite automaton that reads the colors of the play. It depends on the current vertex and the state of the automaton.

Finite memory strategy – example

Muller game:



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

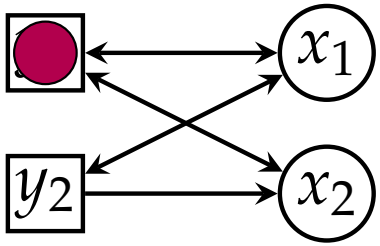


$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

Finite memory strategy – example

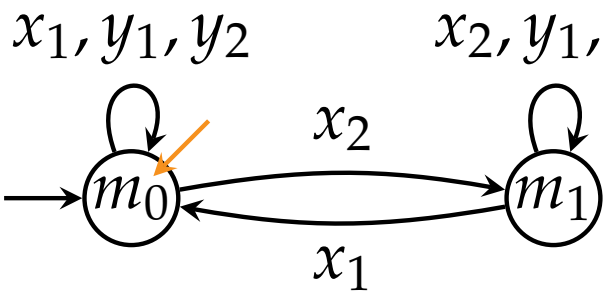
Muller game:



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

y_1

m_0

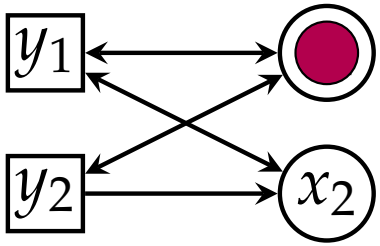


$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

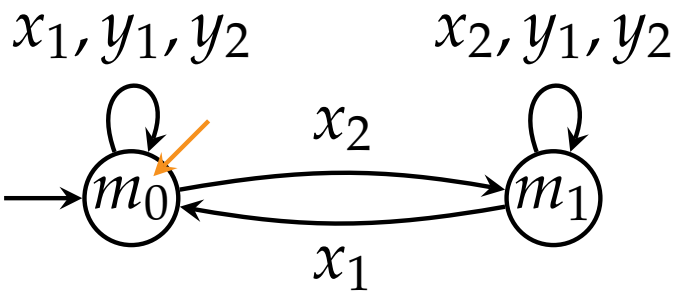
Finite memory strategy – example

Muller game:



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

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

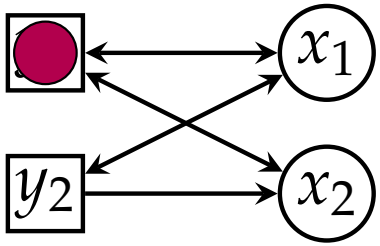


$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

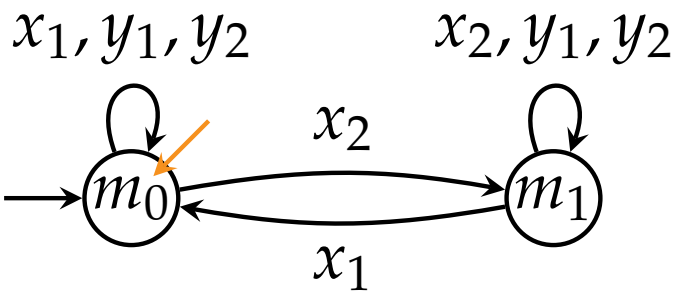
Finite memory strategy – example

Muller game:



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

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

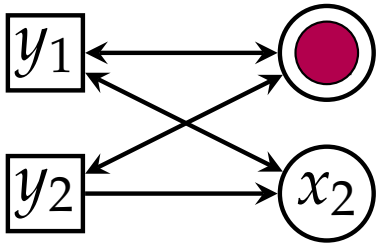


$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

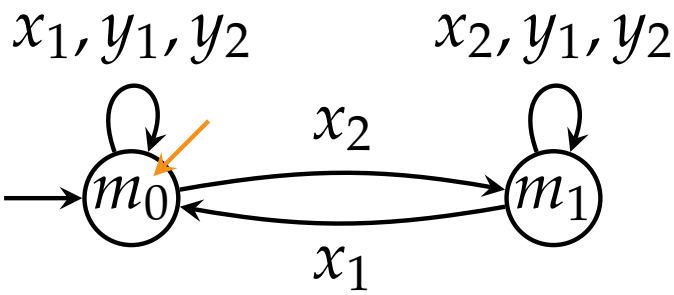
Finite memory strategy – example

Muller game:



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

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

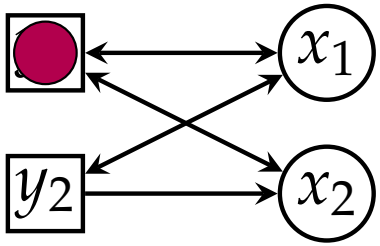


$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

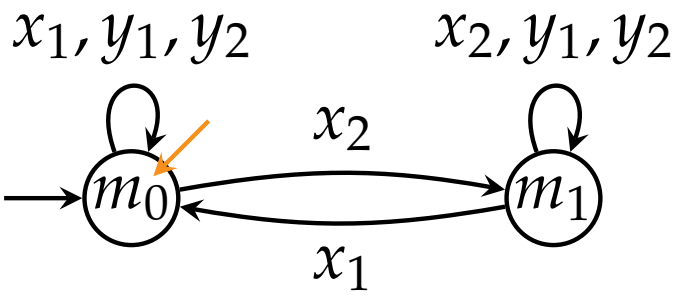
Finite memory strategy – example

Muller game:



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

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

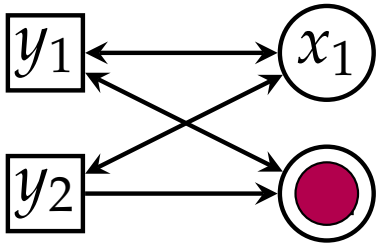


$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

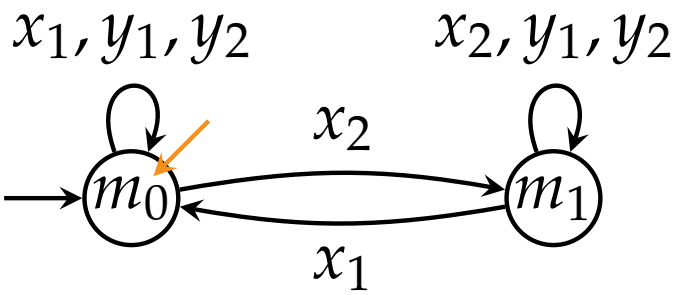
Finite memory strategy – example

Muller game:



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

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

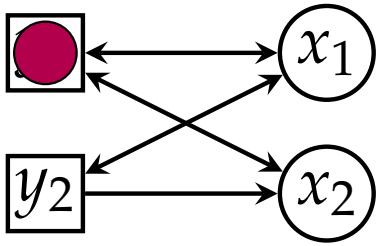


$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

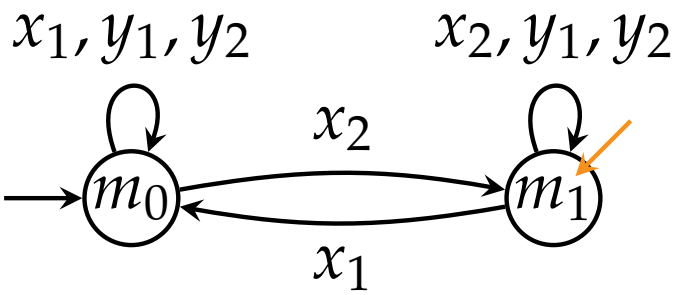
Finite memory strategy – example

Muller game:



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

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

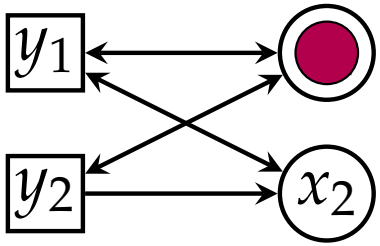


$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

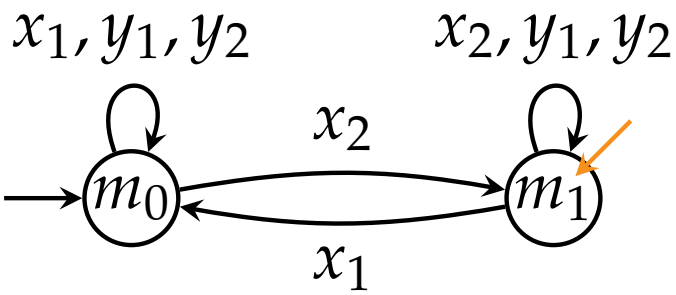
Finite memory strategy – example

Muller game:



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

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

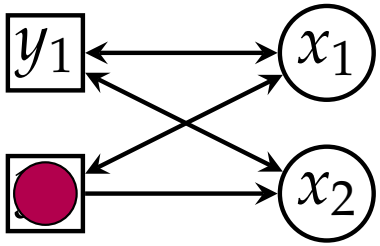


$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

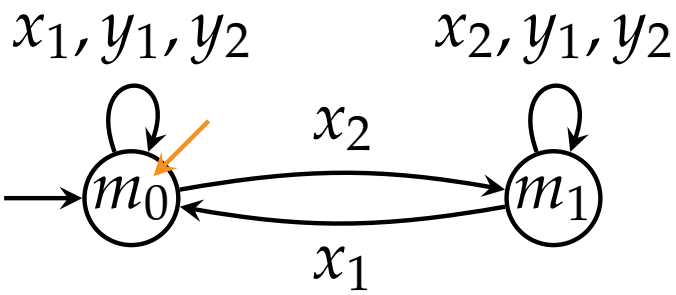
Finite memory strategy – example

Muller game:



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

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

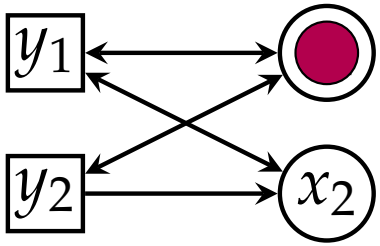


$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

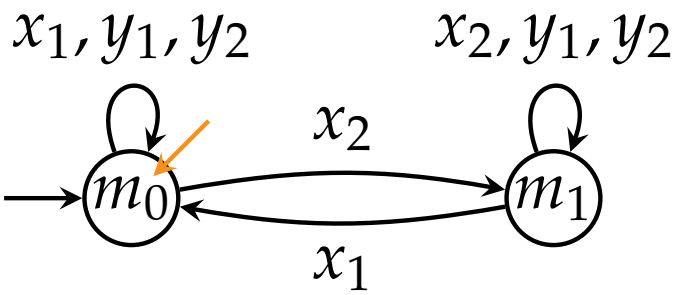
Finite memory strategy – example

Muller game:



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

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

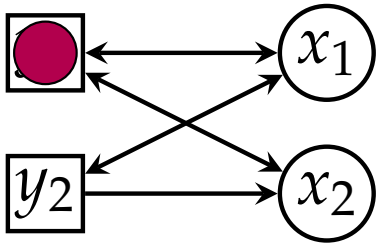


$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

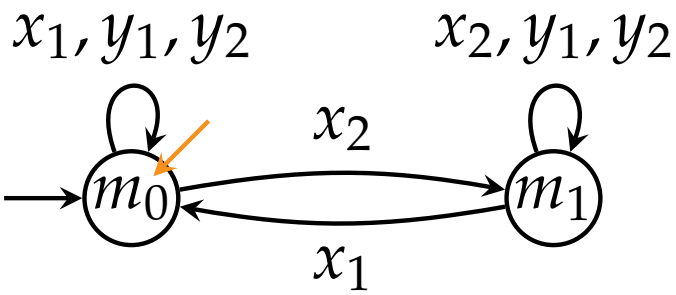
Finite memory strategy – example

Muller game:



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

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



$$\sigma(m_0, x_1) = y_1$$
$$\sigma(m_1, x_1) = y_2$$

"Move to y_2 if the previous x_i was x_2"

Some results (see tutorial of Marcin Jurdziński)

Theorem (Büchi/Landweber'69, Gurevich/Harrington'82, Zielonka'98)

Muller games are determined with finite memory strategies.

Some results (see tutorial of Marcin Jurdziński)

Theorem (Büchi/Landweber'69, Gurevich/Harrington'82, Zielonka'98)

Muller games are determined with finite memory strategies.

Theorem (Emerson/Jutla'88,Mostowski'91)

Parity games are positionally determined.

Some results (see tutorial of Marcin Jurdziński)

Theorem (Büchi/Landweber'69, Gurevich/Harrington'82, Zielonka'98)

Muller games are determined with finite memory strategies.

Theorem (Emerson/Jutla'88,Mostowski'91)

Parity games are positionally determined.

What about other types of winning conditions, e.g.

- Every blue state is followed by a green state, there are at most 3 red states, and if there is no yellow state, then there are infinitely many green ones...

Do we have to design a new algorithm for each winning condition?

1 Introduction

2 Basics on games

3 Transformation of winning conditions

- ω-Automata
- Game reductions
- Logical winning conditions

4 Tree automata

- Complementation
- Emptiness

5 Beyond finite automata

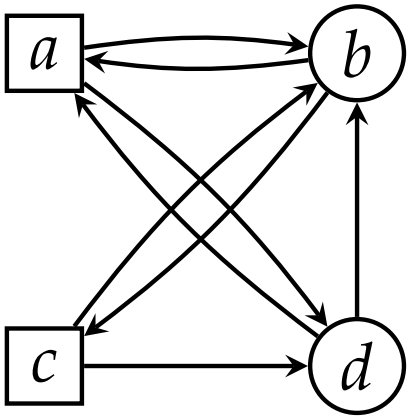
Translation of winning conditions

Goal: general mechanism for transforming winning conditions

Example:

Game graph with colors $C = \{a, b, c, d\}$

Winning condition: Eva wins if the play never matches the regular expression r (over the alphabet C)



$$C^*c(a+b)^*cC^* + C^*d(a+b)^*dC^*$$

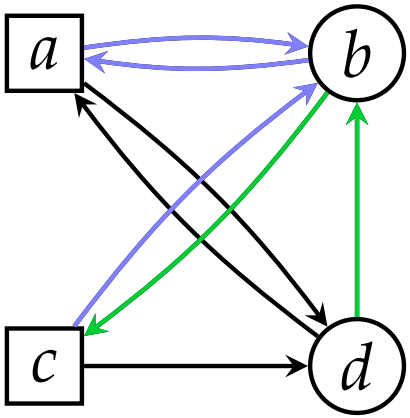
Translation of winning conditions

Goal: general mechanism for transforming winning conditions

Example:

Game graph with colors $C = \{a, b, c, d\}$

Winning condition: Eva wins if the play never matches the regular expression r (over the alphabet C)



$$C^*c(a+b)^*cC^* + C^*d(a+b)^*dC^*$$

strategy with two states of memory

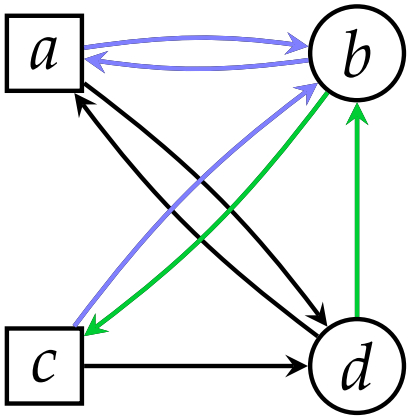
Translation of winning conditions

Goal: general mechanism for transforming winning conditions

Example:

Game graph with colors $C = \{a, b, c, d\}$

Winning condition: Eva wins if the play never matches the regular expression r (over the alphabet C)



$$C^*c(a+b)^*cC^* + C^*d(a+b)^*dC^*$$

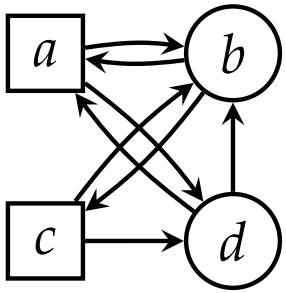
strategy with two states of memory

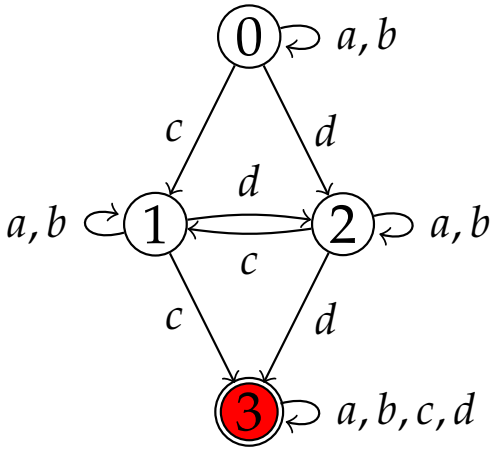
How to solve such games in general?

Safety game

Product of game graph and DFA for r: the automaton reads the play

Eva wins if she can avoid the final states of the DFA.

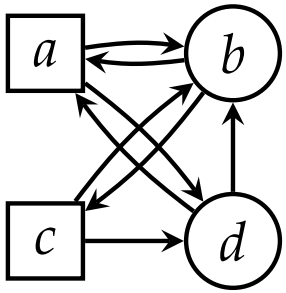


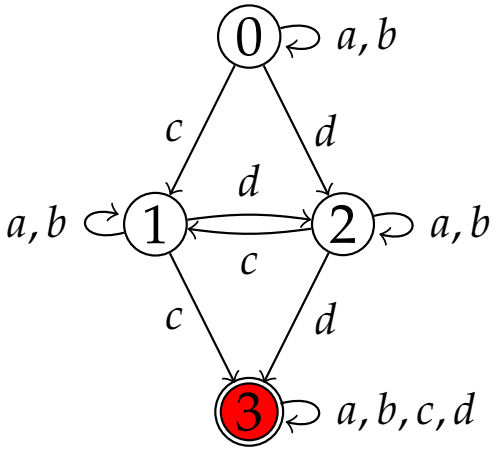


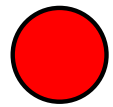
Safety game

Product of game graph and DFA for r: the automaton reads the play

Eva wins if she can avoid the final states of the DFA.



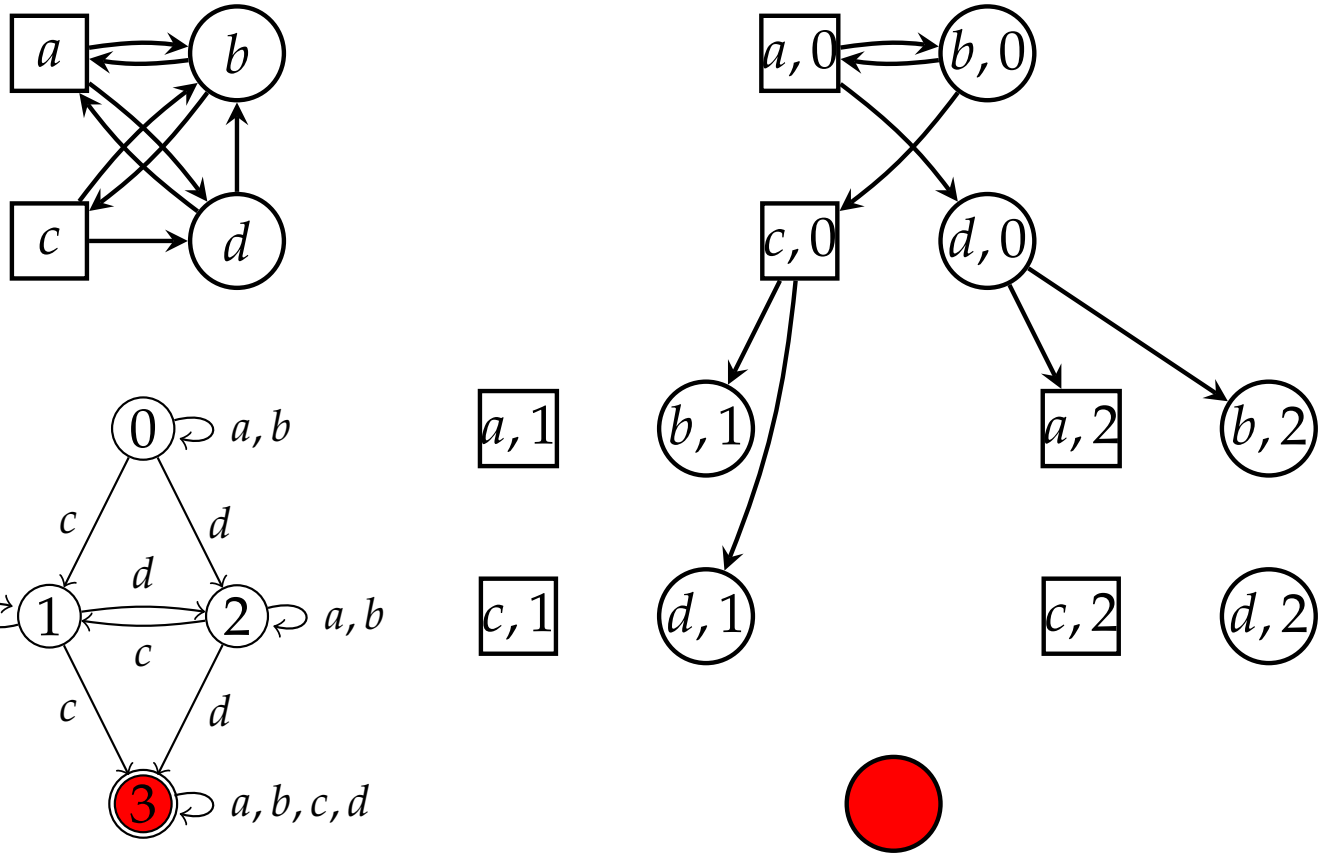




Safety game

Product of game graph and DFA for r: the automaton reads the play

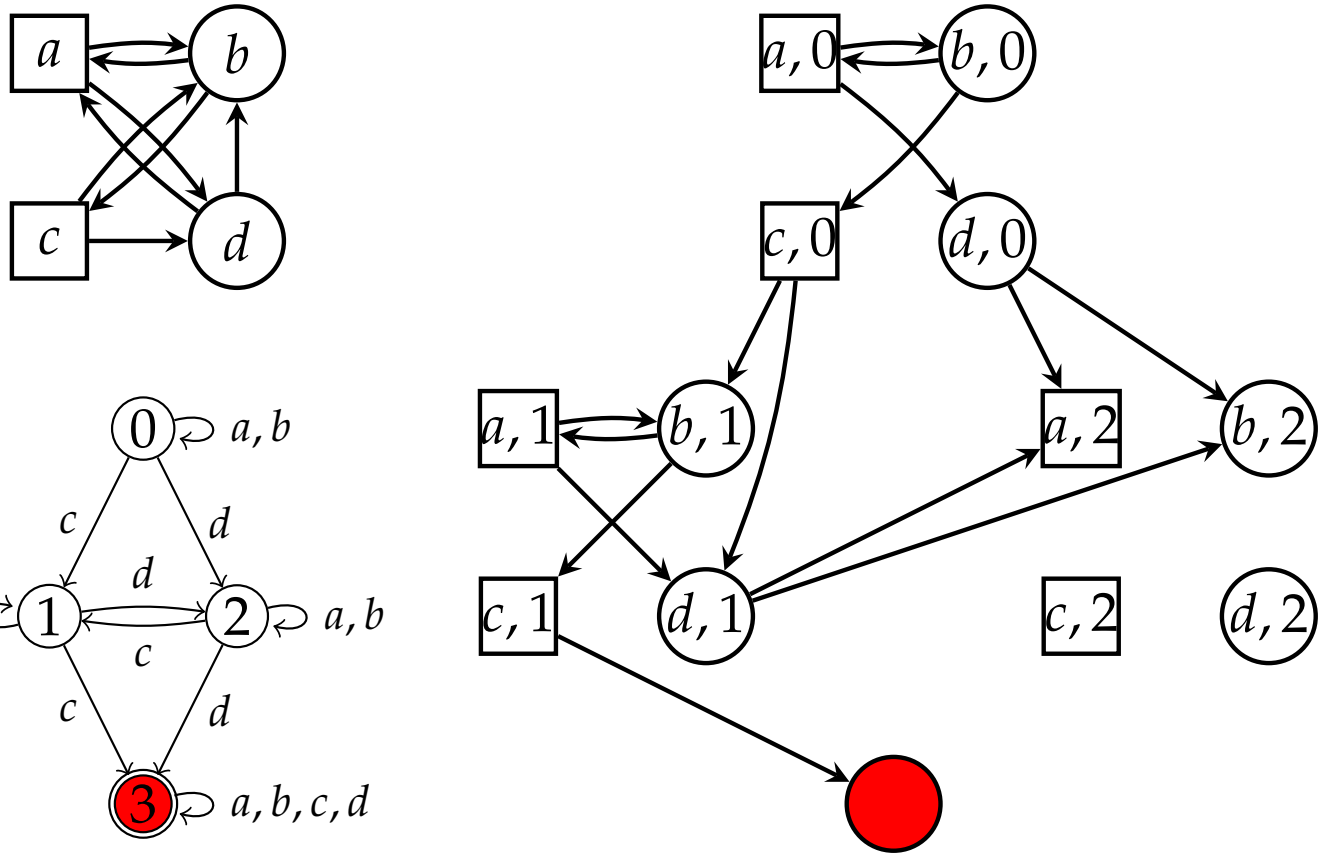
Eva wins if she can avoid the final states of the DFA.



Safety game

Product of game graph and DFA for r: the automaton reads the play

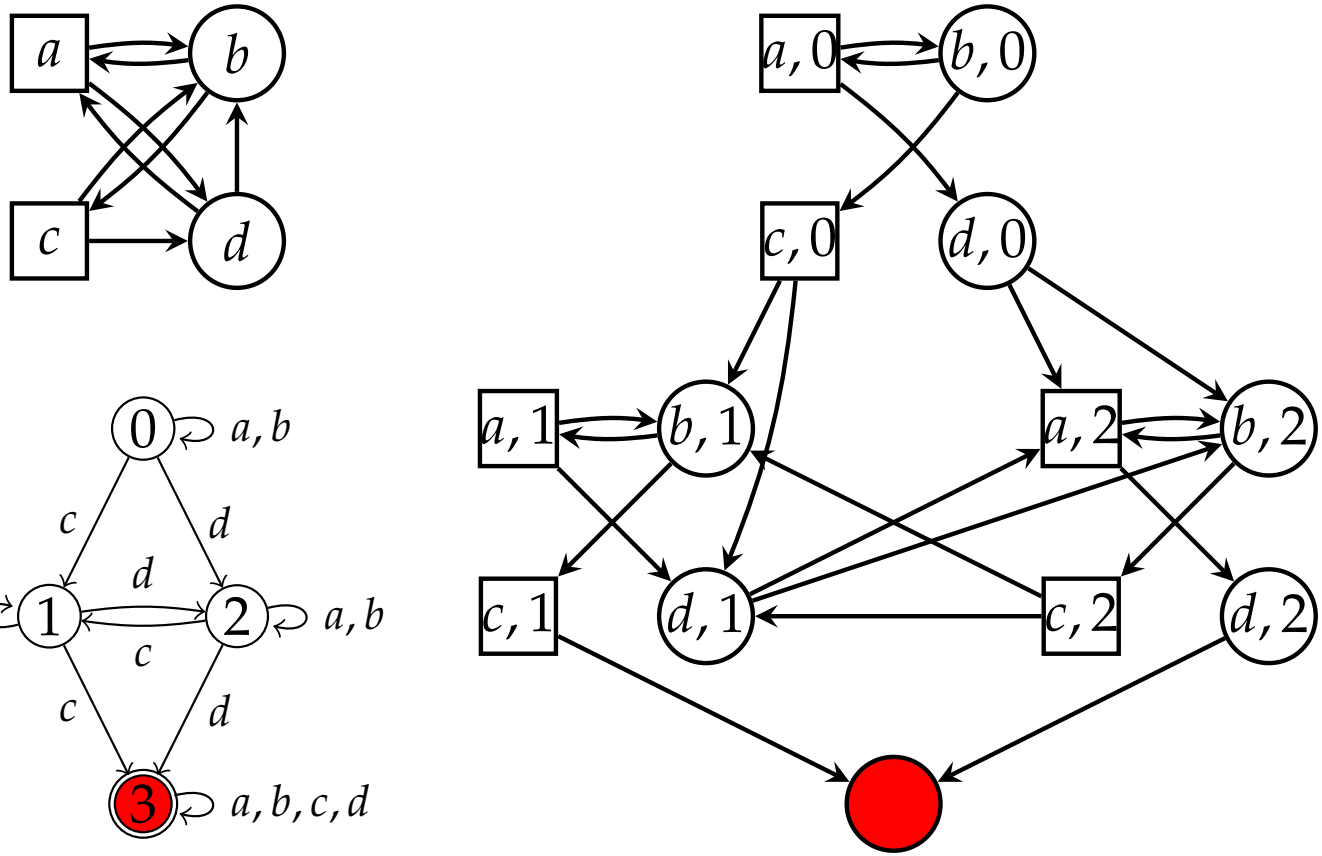
Eva wins if she can avoid the final states of the DFA.



Safety game

Product of game graph and DFA for r: the automaton reads the play

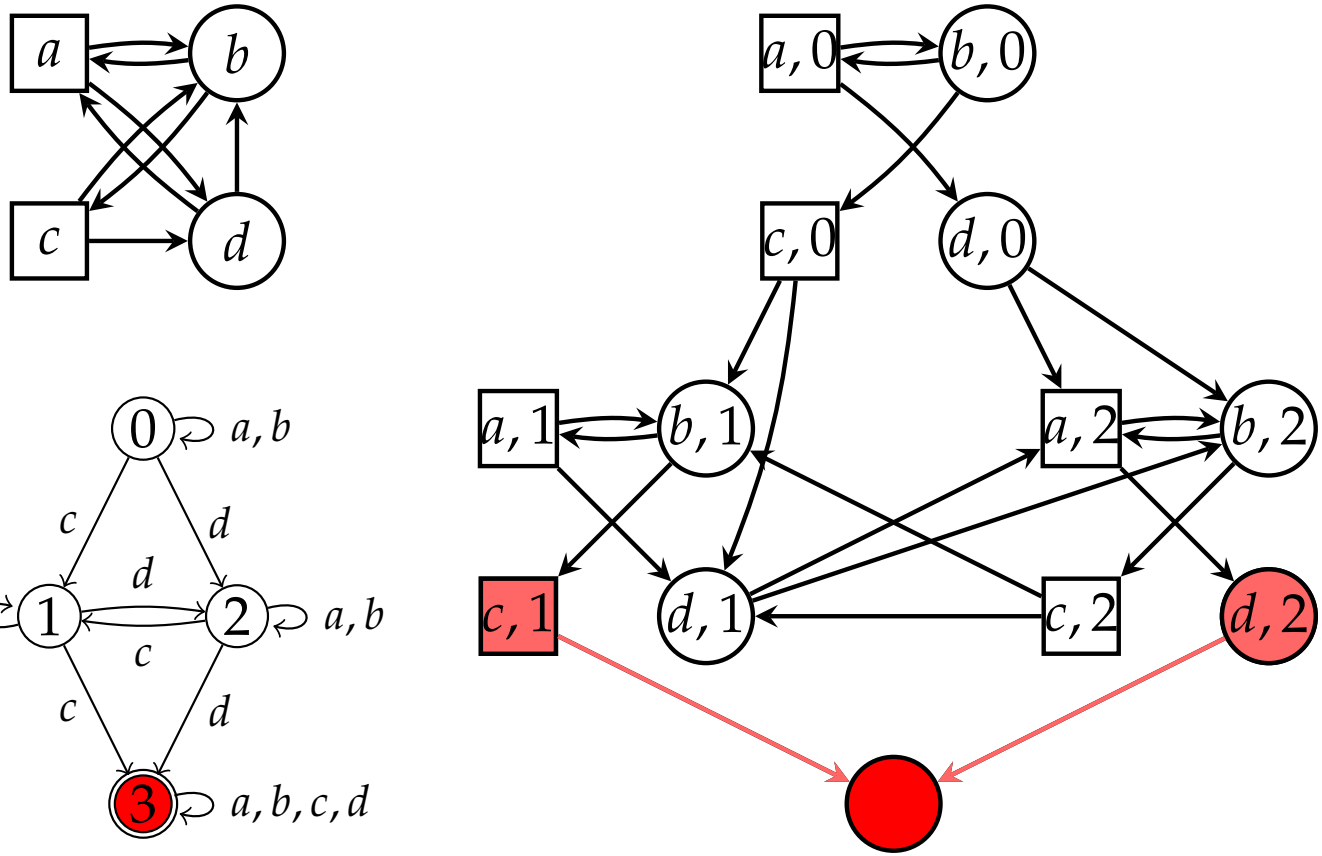
Eva wins if she can avoid the final states of the DFA.



Safety game

Product of game graph and DFA for r: the automaton reads the play

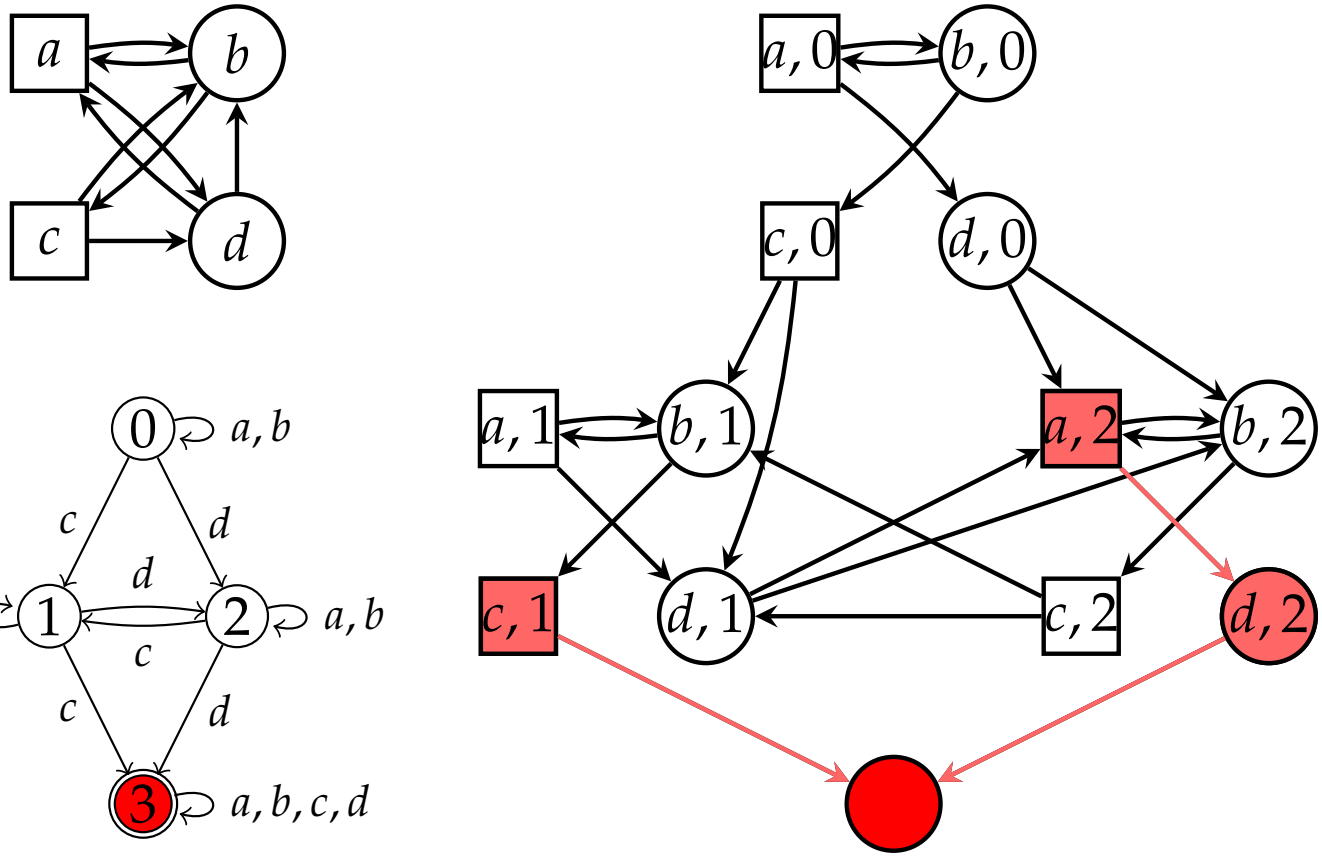
Eva wins if she can avoid the final states of the DFA.



Safety game

Product of game graph and DFA for r: the automaton reads the play

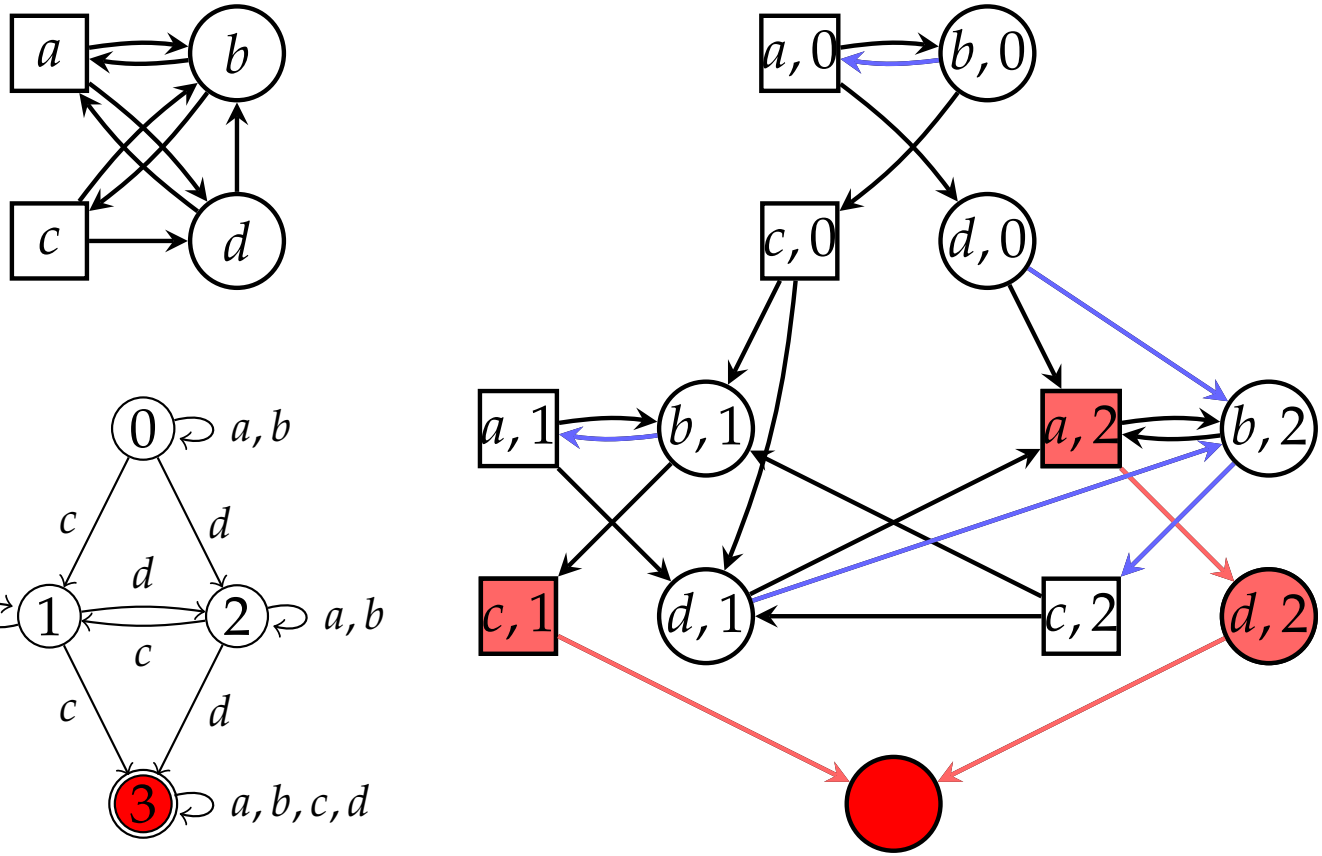
Eva wins if she can avoid the final states of the DFA.



Safety game

Product of game graph and DFA for r: the automaton reads the play

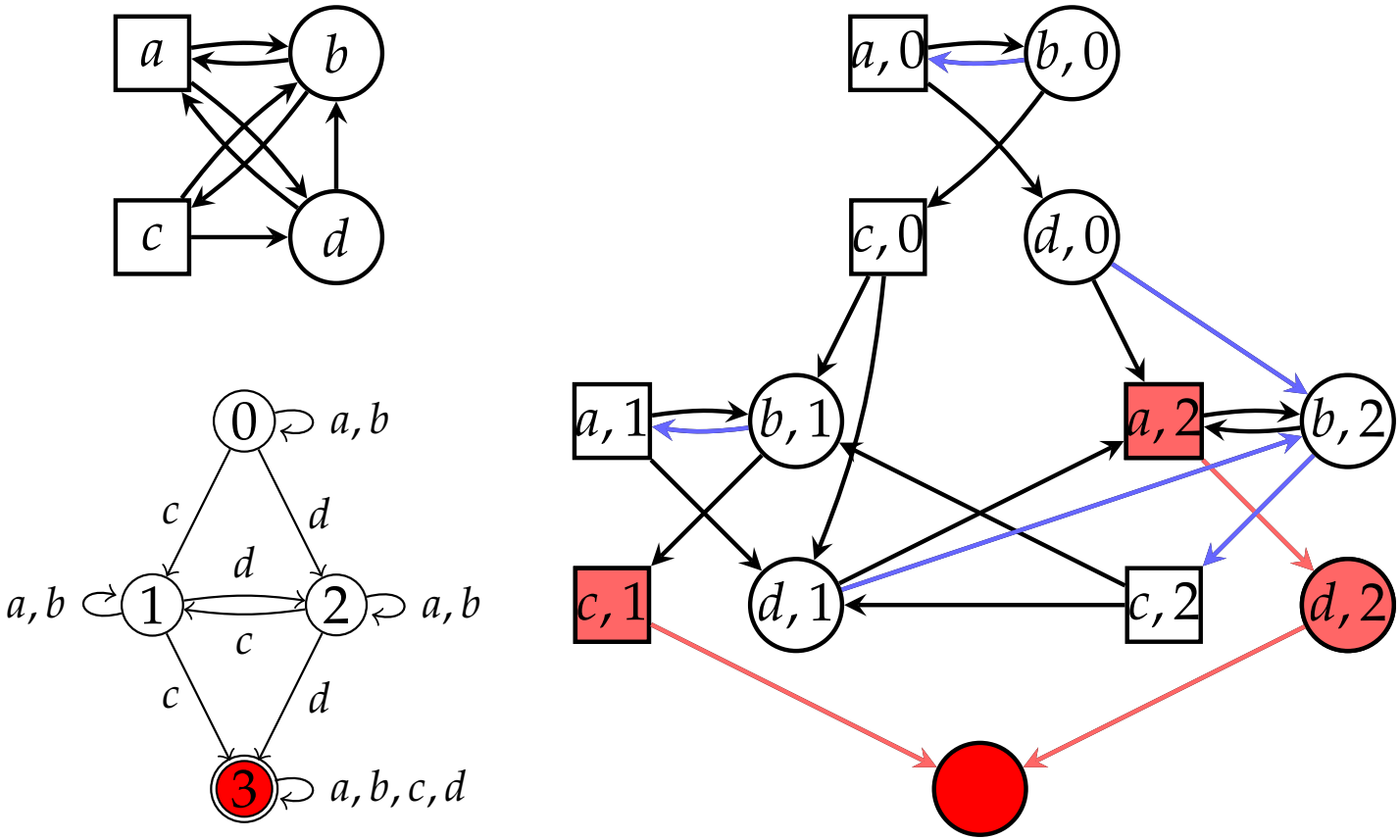
Eva wins if she can avoid the final states of the DFA.



Safety game

Product of game graph and DFA for r: the automaton reads the play

Eva wins if she can avoid the final states of the DFA.



Strategy for Eva with three memory states.

Summary of the method

- Transform the regular expression into a DFA.
- Take the product with the game graph.
- The resulting game is a safety game: Eva wins if she can avoid the set R of red states.
- Solve the safety game by an attractor construction:
  - Compute stepwise the set $Attr_A(R)$ of nodes from where Adam can force a visit to a red state.
  - For the nodes outside of $Attr_A(R)$ Eva has a strategy to stay outside (by the definition of $Attr_A(R)$).
- Translate the strategy for Eva (or for Adam) back to the original game, using the DFA as memory.

Infinitary conditions

- For the above example we used a translation from regular expressions to deterministic finite automata (DFAs).
- For infinitary conditions (something happens infinitely/finitely often) standard DFAs are not enough.
- To treat such conditions we can use ω-automata.

ω-Automata

Büchi automata

An **ω-automaton** is of the form $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, Acc)$, where $Q, \Sigma, q_{in}, \Delta$ are as for standard finite automata, and Acc defines the acceptance condition.

Acceptance conditions:

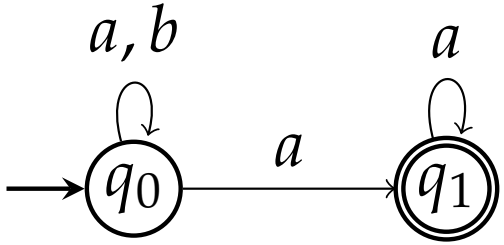
- **Büchi automata**: Acc given as set $F \subseteq Q$ of final states.
  A run is accepting if it contains infinitely often a state from F
- **Parity automata**: Acc given as priority mapping $pri : Q \to \mathbb{N}$
  A run is accepting if the maximal priority appearing infinitely often is even.

Deterministic automata: as usual (at most one transition per state and letter)

Basic examples

$\Sigma = \{a, b\}$

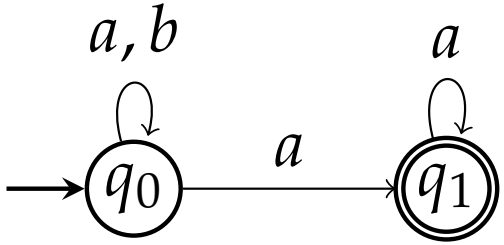
- A nondeterministic Büchi automaton for “finitely many b”:



Basic examples

$\Sigma = \{a, b\}$

- A nondeterministic Büchi automaton for “finitely many b”:

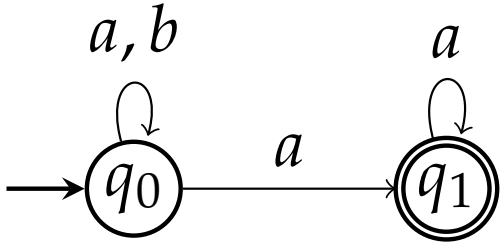


There is no deterministic Büchi automaton for this language

Basic examples

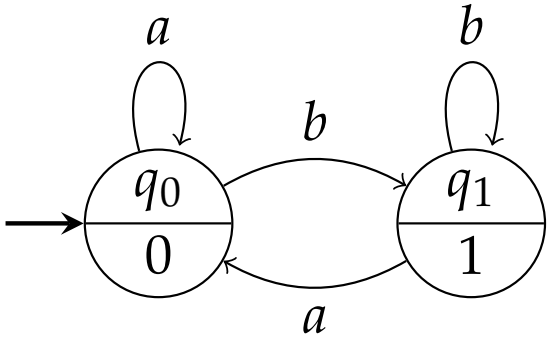
$\Sigma = \{a, b\}$

- A nondeterministic Büchi automaton for "finitely many b":



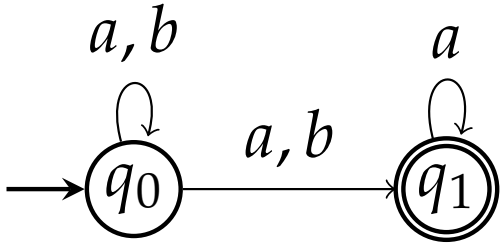
There is no deterministic Büchi automaton for this language

- A deterministic parity automaton for the same language:



The determinization problem

- Classical subset construction fails

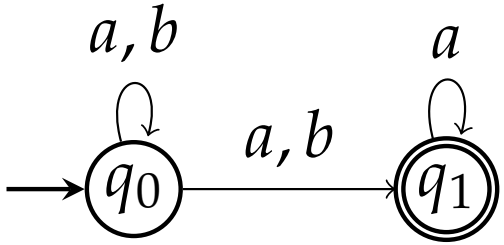


$aaaaaa\cdots$ and $abababab\cdots$ induce the same sequence of sets:

$$\{q_0\} \xrightarrow[a]{a} \{q_0, q_1\} \xrightarrow[b]{a} \{q_0, q_1\} \xrightarrow[a]{a} \{q_0, q_1\} \xrightarrow[b]{a} \{q_0, q_1\} \cdots$$

The determinization problem

- Classical subset construction fails



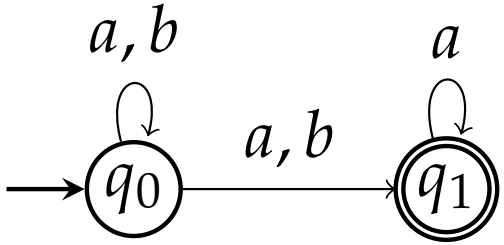
$aaaaaa\cdots$ and $abababab\cdots$ induce the same sequence of sets:

$$\{q_0\} \xrightarrow[a]{a} \{q_0, q_1\} \xrightarrow[b]{a} \{q_0, q_1\} \xrightarrow[a]{a} \{q_0, q_1\} \xrightarrow[b]{a} \{q_0, q_1\} \cdots$$

- A deterministic automaton has to store more information on the possible runs of the Büchi automaton: Besides the states reached some information on the visits to final states are is necessary.

The determinization problem

- Classical subset construction fails



$aaaaaa\cdots$ and $abababab\cdots$ induce the same sequence of sets:

$$\{q_0\} \xrightarrow[a]{a} \{q_0, q_1\} \xrightarrow[b]{a} \{q_0, q_1\} \xrightarrow[a]{a} \{q_0, q_1\} \xrightarrow[b]{a} \{q_0, q_1\} \cdots$$

- A deterministic automaton has to store more information on the possible runs of the Büchi automaton: Besides the states reached some information on the visits to final states are is necessary.
- The details are rather technical...

Determinization

Theorem (McNaughton'66,Safra'88)

For each nondeterministic Büchi automaton with n states there is an equivalent deterministic parity automaton with $2^{\mathcal{O}(n \log n)}$ states.

Determinization

Theorem (McNaughton'66,Safra'88)

For each nondeterministic Büchi automaton with n states there is an equivalent deterministic parity automaton with $2^{\mathcal{O}(n \log n)}$ states.

Deterministic parity automata are easy to complement: increase all priorities by 1.

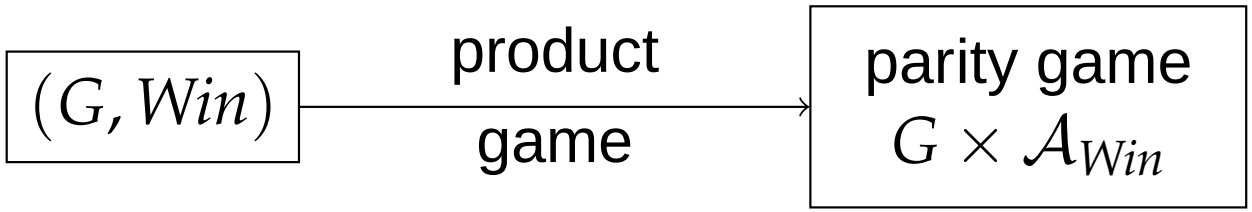
Corollary

The class of regular ω-languages is closed under complementation.

Game reductions

Game reductions

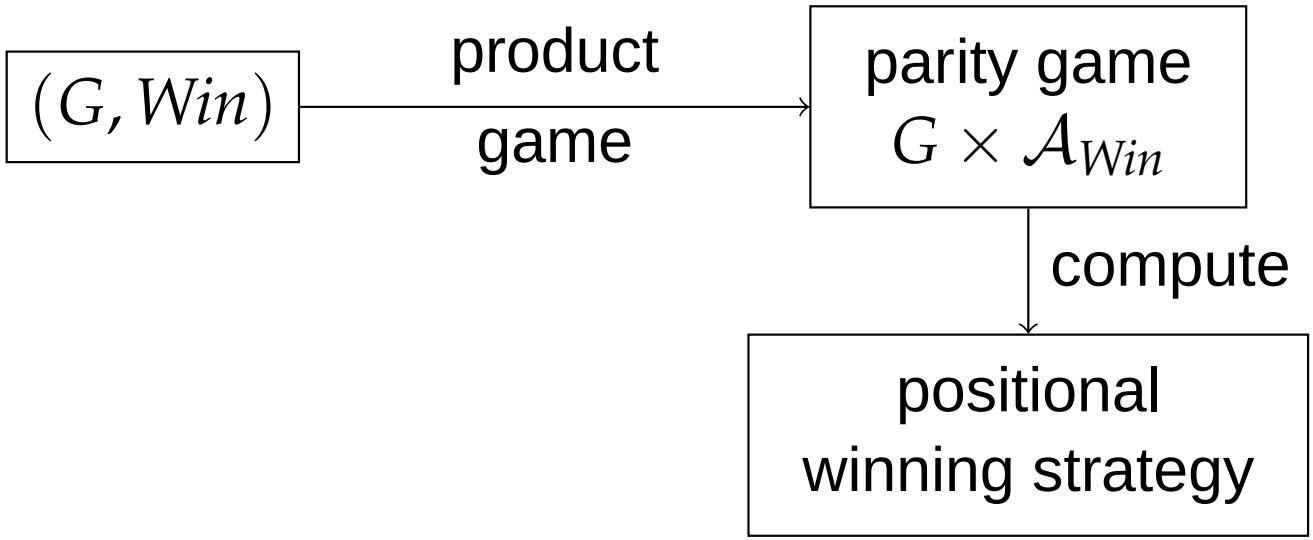
We use the following scheme:



- $\mathcal{A}_{Win}$ is a deterministic parity automaton for Win
- It can be seen as a transducer that transforms sequences from C^ω into sequences of priorities

Game reductions

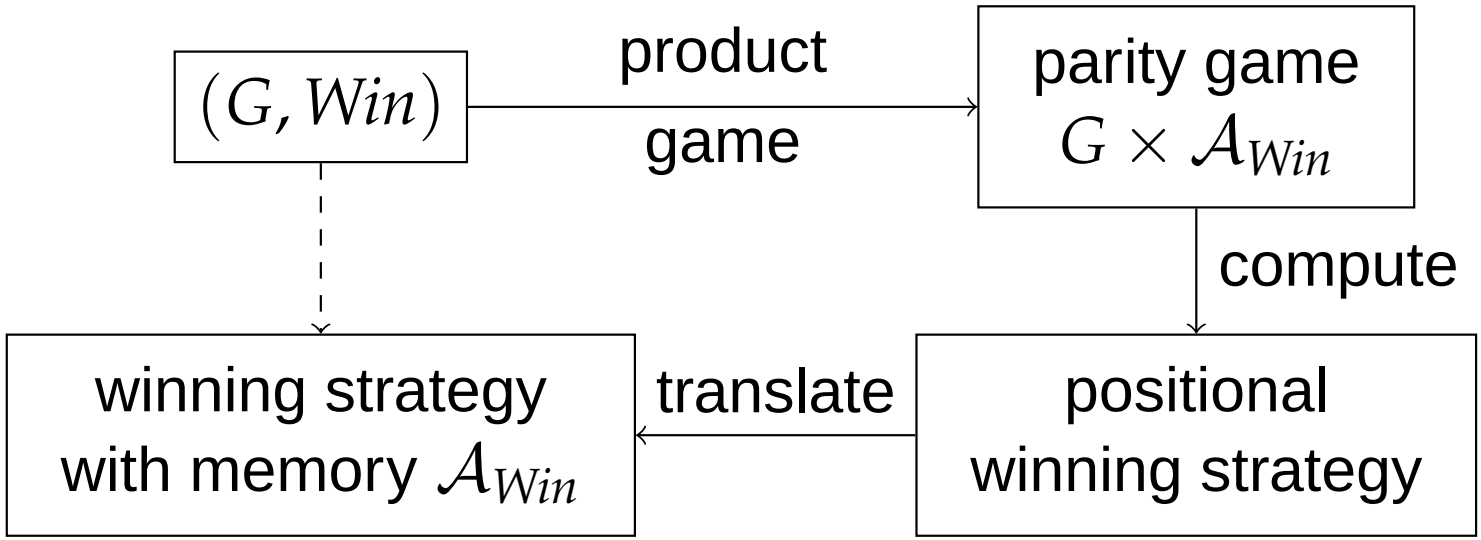
We use the following scheme:



- $\mathcal{A}_{Win}$ is a deterministic parity automaton for Win
- It can be seen as a transducer that transforms sequences from C^ω into sequences of priorities

Game reductions

We use the following scheme:



- $\mathcal{A}_{Win}$ is a deterministic parity automaton for Win
- It can be seen as a transducer that transforms sequences from C^ω into sequences of priorities

Example: Muller to Parity

- There is a direct algorithm for solving Muller games.
- For illustrating the concept of game reduction we provide a translation of Muller games to parity games.
- For this purpose we construct a deterministic parity automaton that
  - reads sequences α from C^ω and
  - accepts if α satisfies the given Muller condition
- The construction is based on “latest appearance records”.

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

$$d \quad b \quad b \quad d \quad c \quad b \quad a \quad b \quad a \quad c \quad b \quad a \quad b \quad a \quad c$$

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c

a
b
c
$\underline{d}$

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c

a	d
b	a
c	b
$\underline{d}$	$\underline{c}$

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

$$d\ b\ b\ d\ c\ b\ a\ b\ a\ c\ b\ a\ b\ a\ c$$

a	d	b
b	a	d
c	b	$\underline{a}$
$\underline{d}$	$\underline{c}$	c

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

$$\begin{array}{cccccccccccccccc} d & b & b & d & c & b & a & b & a & c & b & a & b & a & c \\ \hline a & d & b & \underline{b} \\ b & a & d & d \\ c & b & \underline{a} & a \\ \underline{d} & \underline{c} & c & c \end{array}$$

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d										
b	a	d	d	$\underline{b}$										
c	b	$\underline{a}$	a	a										
$\underline{d}$	$\underline{c}$	c	c	c										

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

$d\ b\ b\ d\ c\ b\ a\ b\ a\ c\ b\ a\ b\ a\ c$

a	d	b	$\underline{b}$	d	c
b	a	d	d	$\underline{b}$	d
c	b	$\underline{a}$	a	a	b
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d	c	b								
b	a	d	d	$\underline{b}$	d	c								
c	b	$\underline{a}$	a	a	b	$\underline{d}$								
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a								

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d	c	b	a							
b	a	d	d	$\underline{b}$	d	c	b							
c	b	$\underline{a}$	a	a	b	$\underline{d}$	c							
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a	$\underline{d}$							

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d	c	b	a	b
b	a	d	d	$\underline{b}$	d	c	b	$\underline{a}$
c	b	$\underline{a}$	a	a	b	$\underline{d}$	c	c
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a	$\underline{d}$	d

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d	c	b	a	b	a					
b	a	d	d	$\underline{b}$	d	c	b	$\underline{a}$	$\underline{b}$					
c	b	$\underline{a}$	a	a	b	$\underline{d}$	c	c	c					
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a	$\underline{d}$	d	d					

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d	c	b	a	b	a	c				
b	a	d	d	$\underline{b}$	d	c	b	$\underline{a}$	$\underline{b}$	a				
c	b	$\underline{a}$	a	a	b	$\underline{d}$	c	c	c	$\underline{b}$				
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a	$\underline{d}$	d	d	d				

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d	c	b	a	b	a	c	b			
b	a	d	d	$\underline{b}$	d	c	b	$\underline{a}$	$\underline{b}$	a	c			
c	b	$\underline{a}$	a	a	b	$\underline{d}$	c	c	c	$\underline{b}$	$\underline{a}$			
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a	$\underline{d}$	d	d	d	d			

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d	c	b	a	b	a	c	b	a		
b	a	d	d	$\underline{b}$	d	c	b	$\underline{a}$	$\underline{b}$	a	c	b		
c	b	$\underline{a}$	a	a	b	$\underline{d}$	c	c	c	$\underline{b}$	$\underline{a}$	$\underline{c}$		
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a	$\underline{d}$	d	d	d	d	d		

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d	c	b	a	b	a	c	b	a	b	
b	a	d	d	$\underline{b}$	d	c	b	$\underline{a}$	$\underline{b}$	a	c	b	$\underline{a}$	
c	b	$\underline{a}$	a	a	b	$\underline{d}$	c	c	c	$\underline{b}$	$\underline{a}$	$\underline{c}$	c	
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a	$\underline{d}$	d	d	d	d	d	d	

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d	c	b	a	b	a	c	b	a	b	a
b	a	d	d	$\underline{b}$	d	c	b	$\underline{a}$	$\underline{b}$	a	c	b	$\underline{a}$	$\underline{b}$
c	b	$\underline{a}$	a	a	b	$\underline{d}$	c	c	c	$\underline{b}$	$\underline{a}$	$\underline{c}$	c	c
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a	$\underline{d}$	d	d	d	d	d	d	d

- When a color is moved to the front (top), mark its previous position
- This allows to infer from the LAR which colors are visited infinitely often

Latest Appearance Record – Idea

- Recall the order in which the colors (or vertices) of the play appeared (starting with an arbitrary order)
- The colors appearing finitely often will gather at one end

d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d	c	b	a	b	a	c	b	a	b	a	c
b	a	d	d	$\underline{b}$	d	c	b	$\underline{a}$	$\underline{b}$	a	c	b	$\underline{a}$	$\underline{b}$	a
c	b	$\underline{a}$	a	a	b	$\underline{d}$	c	c	c	$\underline{b}$	$\underline{a}$	$\underline{c}$	c	c	$\underline{b}$
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a	$\underline{d}$	d	d	d	d	d	d	d	d

- When a color is moved to the front (top), mark its previous position
- 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.

$$LAR(C) = \left\{ [d_1 \cdots d_n, h] \;\middle|\; \begin{array}{l} d_i \in C, d_i \neq d_j \text{ for all } i \neq j, \\ \text{and } 1 \leq h \leq n \end{array} \right\}$$

- LAR update:

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

  for the unique i with $d = d_i$
- This defines the states (the LARs) and the transition structure (LAR update) of the deterministic parity automaton.

Assigning Priorities

- Priority depends on the size of the part of the LAR that has changed in the last transition.
- 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	$\underline{b}$	d	c	b	a	b	a	c	b	a	b	a	c
b	a	d	d	$\underline{b}$	d	c	b	$\underline{a}$	$\underline{b}$	a	c	b	$\underline{a}$	$\underline{b}$	a
c	b	$\underline{a}$	a	a	b	$\underline{d}$	c	c	c	$\underline{b}$	$\underline{a}$	$\underline{c}$	c	c	$\underline{b}$
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a	$\underline{d}$	d	d	d	d	d	d	d	d
7	7	5	1	4	7	5	7	3	3	6	6	6	3	3	6

The LAR automaton

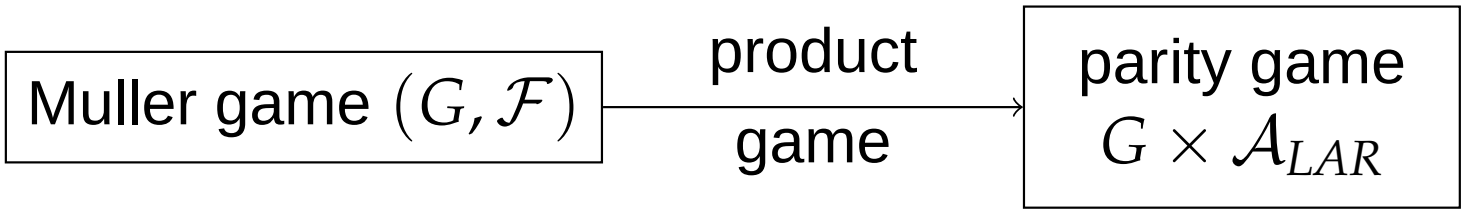
- Let $\mathcal{F}$ be a Muller condition over $|C| = n$ colors.
- Define the deterministic parity automaton $\mathcal{A}_{LAR} = (LAR(C), C, q_{in}, \delta_{LAR}, c_{LAR})$ with

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

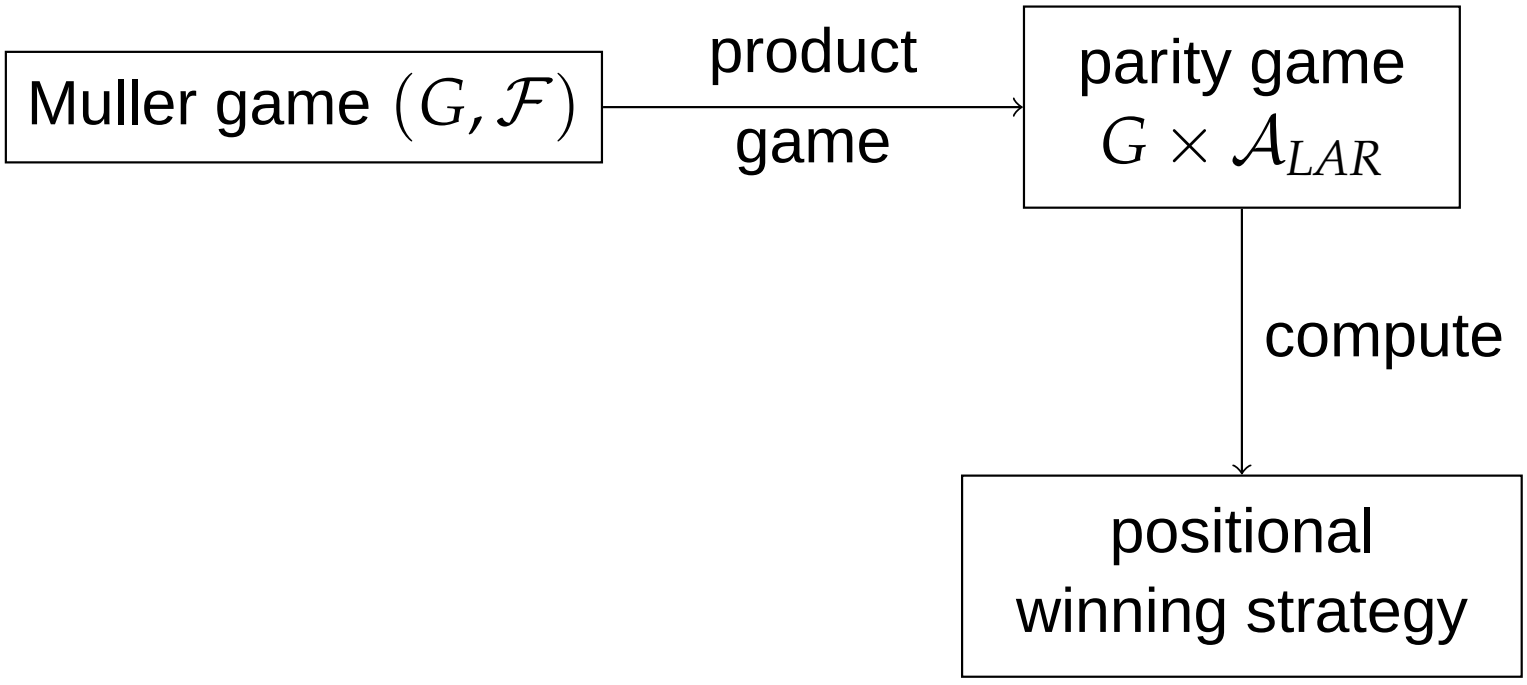
Theorem

For a Muller condition $\mathcal{F}$ over C the corresponding deterministic parity automaton $\mathcal{A}_{LAR}$ accepts precisely those $\alpha \in C^\omega$ that satisfy the Muller condition $\mathcal{F}$.

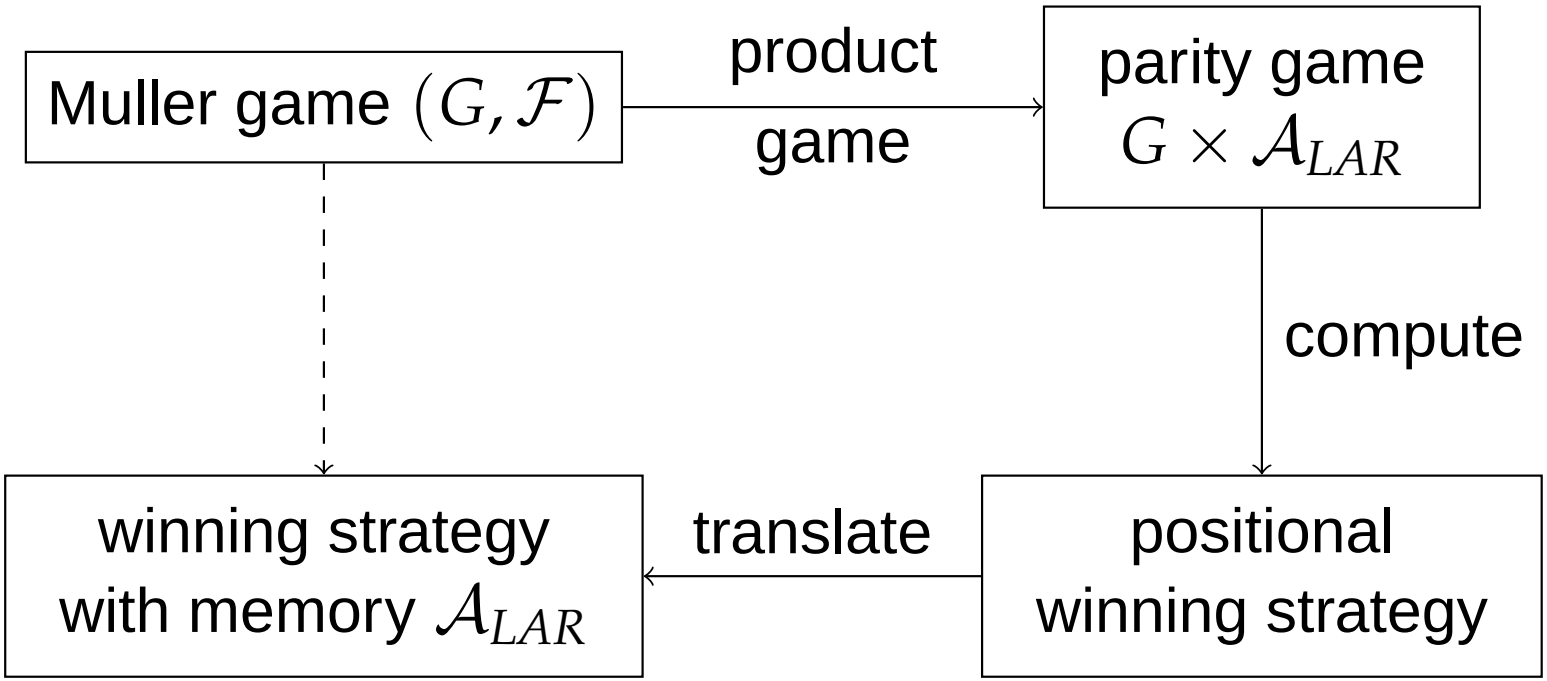
From Muller to Parity Games



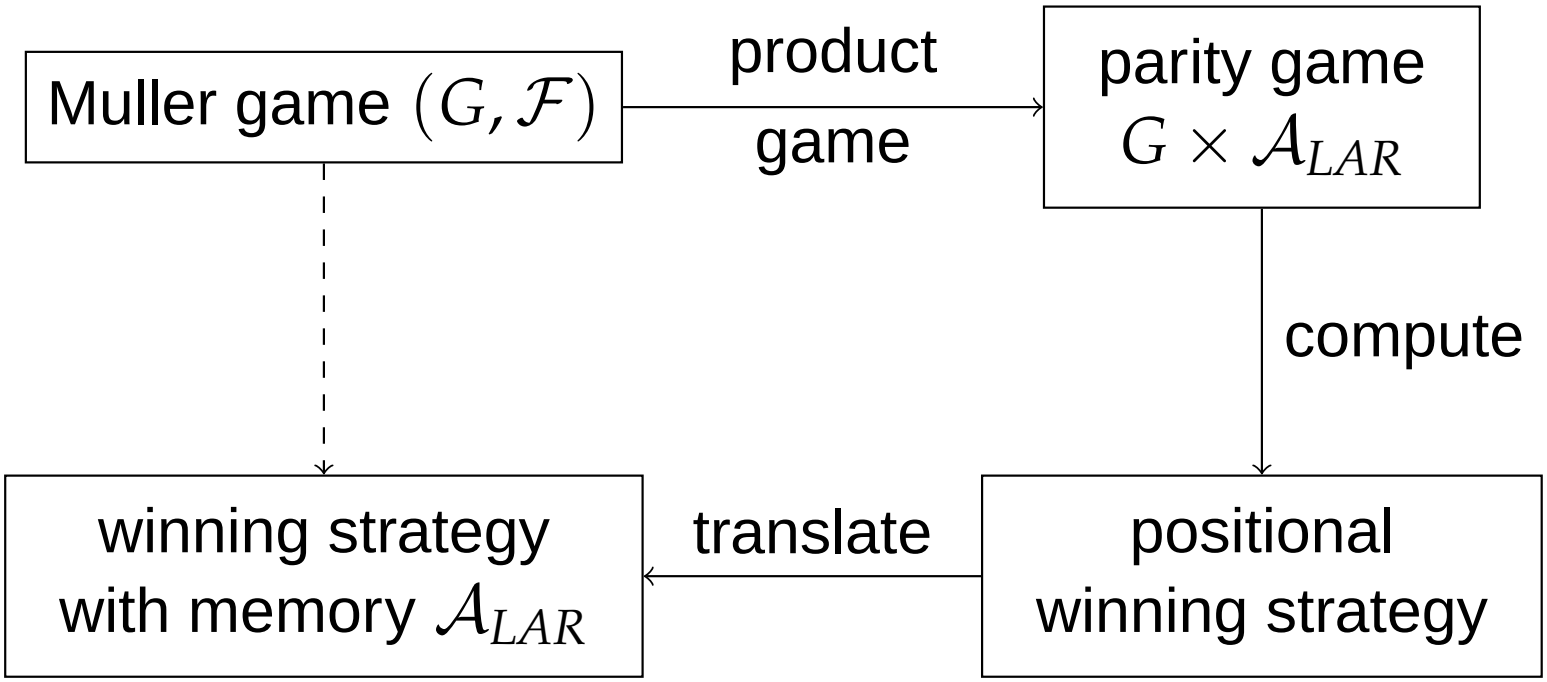
From Muller to Parity Games



From Muller to Parity Games



From Muller to Parity Games



Theorem (Gurevich/Harrington'82)

Muller games are determined with the LAR automaton as memory.

Logical winning conditions

Logic for infinite words

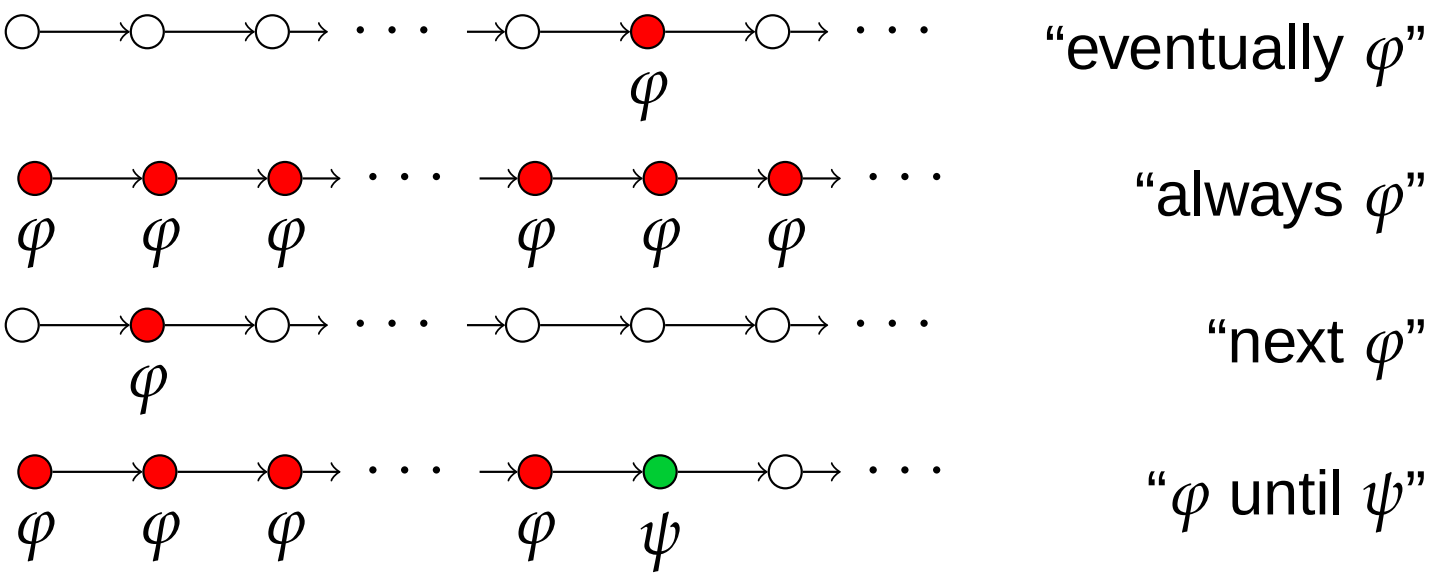
- Used for specifying properties of system executions
- In practice usually temporal logics are used
- From a theoretical point of view and as reference for expressive power predicate logic is interesting
- Here we consider
  - the linear temporal logic LTL and
  - monadic second-order logic over infinite words, called S1S.

Linear temporal logic – LTL

LTL formulas: Build over a set $P = \{p_1, \ldots, p_n\}$ of atomic propositions.

Models: Infinite sequences of vectors of size n. Entry i of a vector codes truth value of p_i ($1 =$ true, $0 =$ false).

- Atomic formulas: p_i (p_i is true in the first position)
- Boolean combinations
- Temporal operators:



Examples

$p_1 \wedge X \neg p_2$

$$\binom{1}{1}\binom{1}{0}\binom{0}{1}\binom{0}{0}\ldots$$

Examples

$p_1 \wedge X \neg p_2$

$$\binom{1}{1}\binom{1}{0}\binom{0}{1}\binom{0}{0}\cdots$$

$\mathsf{G} p_2 \wedge \mathsf{F} p_1$

$$\binom{0}{1}\binom{0}{1}\binom{0}{1}\cdots\binom{0}{1}\binom{1}{1}\binom{0}{1}\binom{0}{1}\binom{1}{1}\cdots$$

Examples

$p_1 \wedge X\neg p_2$

$$\binom{1}{1}\binom{1}{0}\binom{0}{1}\binom{0}{0}\cdots$$

$\mathsf{G}p_2 \wedge \mathsf{F}p_1$

$$\binom{0}{1}\binom{0}{1}\binom{0}{1}\cdots\binom{0}{1}\binom{1}{1}\binom{0}{1}\binom{0}{1}\binom{1}{1}\cdots$$

$\mathsf{F}(p_3 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$

$$\binom{0}{1}{0}\binom{0}{0}{1}\cdots\binom{1}{0}{1}\binom{0}{0}{1}\binom{0}{0}{0}\binom{1}{1}{0}\cdots$$

From LTL to automata

Büchi automaton “guesses” valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

From LTL to automata

Büchi automaton "guesses" valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$							
$\neg p_2$							
$\neg p_2 \mathsf{U} p_1$							
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$							
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$							
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$							

From LTL to automata

Büchi automaton "guesses" valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0						
$\neg p_2$	1						
$\neg p_2 \mathsf{U} p_1$	1						
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0						
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0						
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1						

From LTL to automata

Büchi automaton "guesses" valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0	1					
$\neg p_2$	1	0					
$\neg p_2 \mathsf{U} p_1$	1	0					
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1					
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1					
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1	1					

From LTL to automata

Büchi automaton "guesses" valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0	1	0				
$\neg p_2$	1	0	0				
$\neg p_2 \mathsf{U} p_1$	1	0	1				
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	1				
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	0				
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1	1	1				

From LTL to automata

Büchi automaton "guesses" valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0	1	0	1			
$\neg p_2$	1	0	0	1			
$\neg p_2 \mathsf{U} p_1$	1	0	1	1			
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	1	1			
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	0	1			
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1	1	1	1			

From LTL to automata

Büchi automaton "guesses" valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0	1	0	1	0		
$\neg p_2$	1	0	0	1	1		
$\neg p_2 \mathsf{U} p_1$	1	0	1	1	1		
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	1	1	0		
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	0	1	0		
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1	1	1	1	$\cdot$		

From LTL to automata

Büchi automaton "guesses" valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0	1	0	1	0	1	$\cdots$
$\neg p_2$	1	0	0	1	1	0	$\cdots$
$\neg p_2 \mathsf{U} p_1$	1	0	1	1	1	0	$\cdots$
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	1	1	0	$\cdot$	$\cdots$
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	0	1	0	$\cdot$	$\cdots$
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1	1	1	1	$\cdot$	$\cdot$	$\cdots$

Games with LTL conditions

Theorem

- *For each LTL formula φ one can construct an equivalent Büchi automaton $\mathcal{A}_\varphi$ of size exponential in φ. (Vardi/Wolper'86)*
- *For each LTL formula φ one can construct an equivalent deterministic parity automaton of size doubly exponential in φ. (determinization theorem)*
- *Finite games (G, φ) with a winning condition given by an LTL formula can be solved in doubly exponential time. (game reduction)*

S1S

"Second-order theory of one successor":

- Monadic second-order logic over the structure $(\mathbb{N}, +1)$

  $0 \to 1 \to 2 \to \cdots \to i \longrightarrow i+1 \longrightarrow i+2 \longrightarrow \cdots$

- Extension of first-order logic over $(\mathbb{N}, +1)$ by quantifiers for sets of positions (numbers).

S1S

"Second-order theory of one successor":

- Monadic second-order logic over the structure $(\mathbb{N}, +1)$

  $0 \to 1 \to 2 \to \cdots \to i \longrightarrow i+1 \longrightarrow i+2 \longrightarrow \cdots$

- Extension of first-order logic over $(\mathbb{N}, +1)$ by quantifiers for sets of positions (numbers).

- Example: $\varphi(X) = \exists Y \Big(\quad 0 \in Y \wedge$
  $\forall x (x \in Y \leftrightarrow x+1 \notin Y) \wedge$
  $\forall x (x \in X \to x \in Y) \Big)$

  φ defines the set of all ω-words over $\{0,1\}$ such that 1 can only occur on even positions.

S1S

"Second-order theory of one successor":

- Monadic second-order logic over the structure $(\mathbb{N}, +1)$

  $0 \to 1 \to 2 \to \cdots \to i \longrightarrow i+1 \longrightarrow i+2 \longrightarrow \cdots$

- Extension of first-order logic over $(\mathbb{N}, +1)$ by quantifiers for sets of positions (numbers).

- Example: $\varphi(X) = \exists Y \Big(\quad 0 \in Y \wedge$
  $\forall x (x \in Y \leftrightarrow x+1 \notin Y) \wedge$
  $\forall x (x \in X \to x \in Y) \Big)$

  φ defines the set of all ω-words over $\{0,1\}$ such that 1 can only occur on even positions.

- In general we consider formulas $\varphi(X_1, \ldots, X_n)$ with n free set variables defining ω-languages over $\{0,1\}^n$.

Equivalence to automata

Theorem (Büchi'62)

A language $L \subseteq (\{0,1\}^n)^\omega$ is definable by an S1S formula iff it can be accepted by a nondeterministic Büchi automaton.

Proof:

- From formulas to automata use an inductive translation, based on the closure properties of automata.
- From automata to formulas: Write a formula that describes the existence of an accepting run.
  Sets X_q for each state q code the positions of the run where the automaton is in state q.

Games with S1S conditions

Using determinization of Büchi automata and game reduction:

Theorem (Büchi/Landweber'69)

For finite games (G, φ) with a winning condition given by an S1S formula one can decide the winner and can compute a corresponding winning strategy.

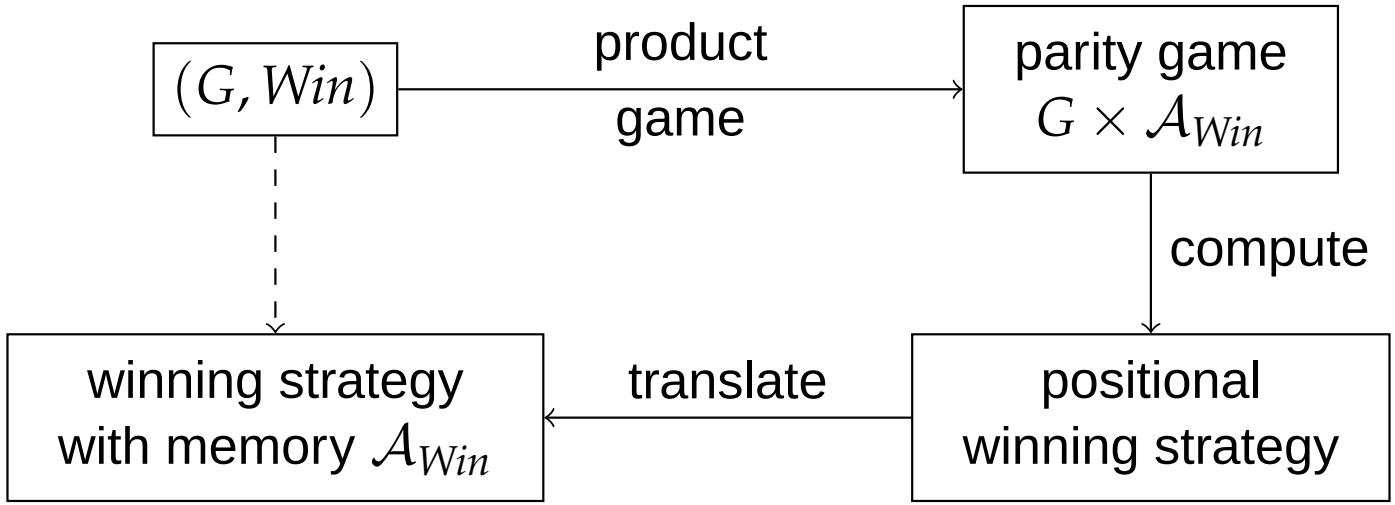
Complexity: The size of the memory required for such a winning strategy cannot be bounded by function

$$\left.\begin{array}{c} 2^{2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}} \end{array}\right\} k$$

for a fixed k.

Summary of this part

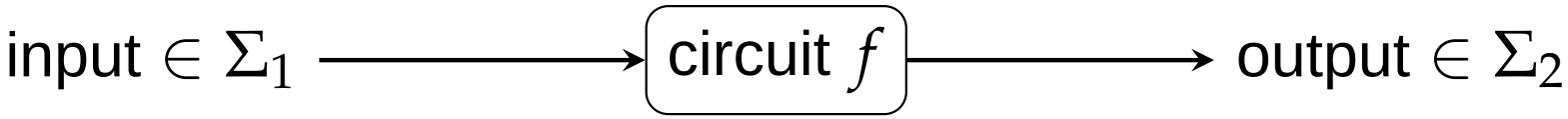
Solving infinite games with the help of automata: Extend the game by running a **deterministic** automaton along the plays.



Examples:

- Muller conditions: LAR construction
- LTL conditions: nondeterministic automaton guessing valuations of subformulas + determinization
- S1S conditions: inductive translation to automata

Solving Church's synthesis problem



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

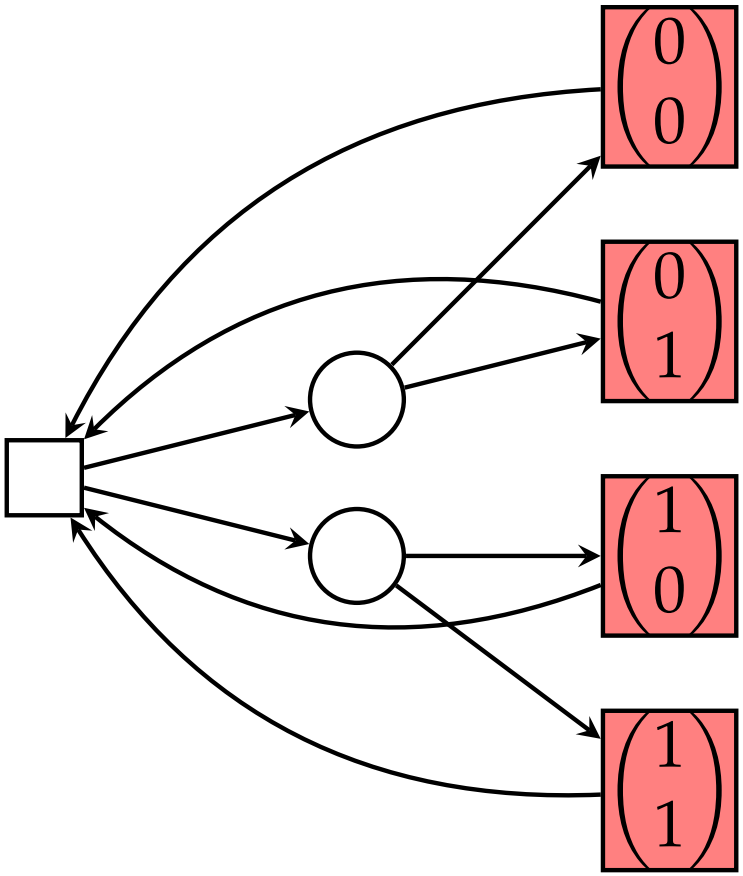
Assume $\Sigma_1 = \Sigma_2 = \{0, 1\}$.

- A computation of f is an infinite sequence of the form

$$\binom{0}{1}\binom{1}{1}\binom{0}{0}\binom{0}{0}\binom{1}{1}\dots$$

  with the input signals in the first row, the output signals in the second row.
- The specification can be written as LTL formula over $\{p_1, p_2\}$ or as S1S formula $\varphi(X_1, X_2)$.

Church's synthesis problem as game



Winning condition: Specification φ rewritten such that it only considers the red vertices.

Finite state winning strategy for Eva gives the desired program for f.

1 Introduction

2 Basics on games

3 Transformation of winning conditions

- ω-Automata
- Game reductions
- Logical winning conditions

4 Tree automata

- Complementation
- Emptiness

5 Beyond finite automata

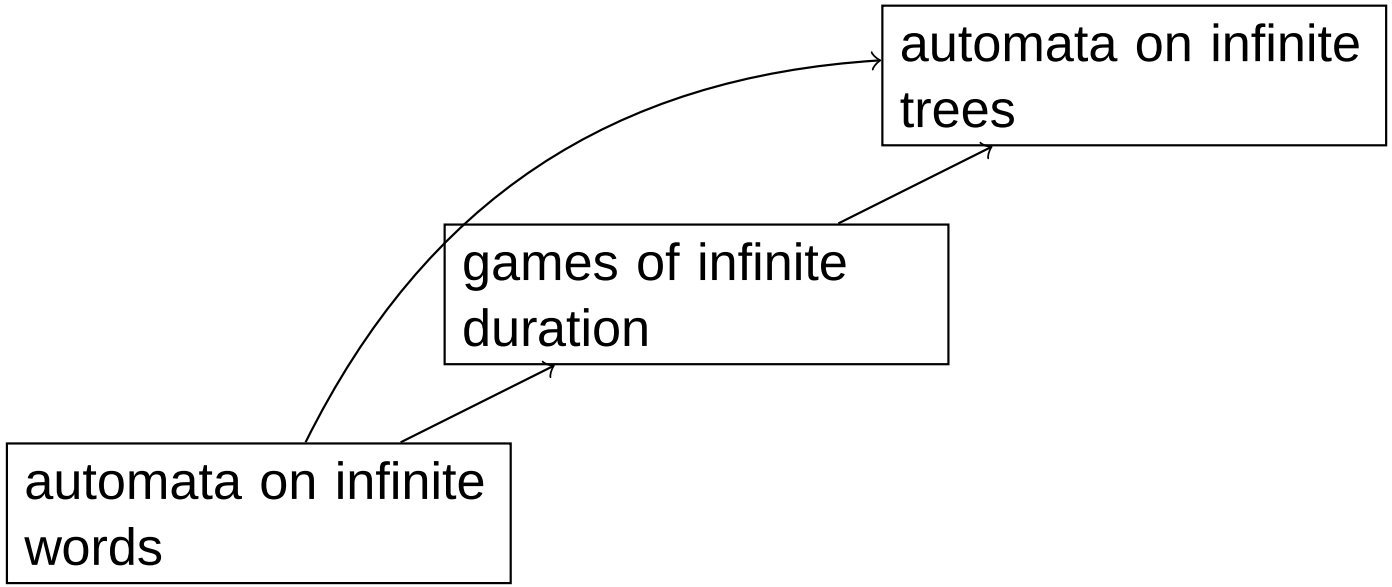
Reminder

Theorem (Emerson/Jutla'88,Mostowski'91)

Parity games are positionally determined.

Theorem (McNaughton'66,Safra'88)

For each nondeterministic Büchi automaton with n states there is an equivalent deterministic parity automaton with $2^{\mathcal{O}(n \log n)}$ states.

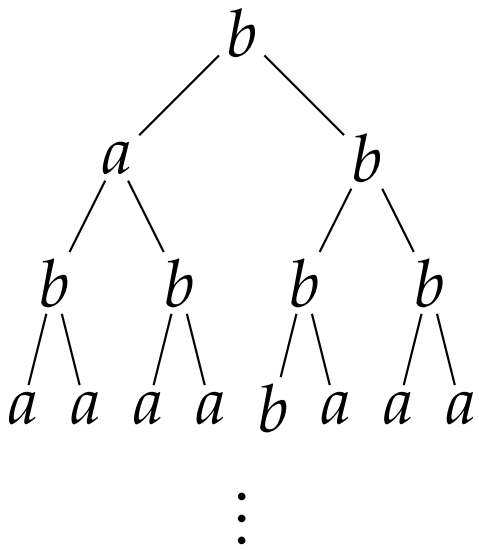


Why infinite trees?

- Discrete systems (circuits, protocols etc.) can be described by transitions graphs.
- The possible behavior of such a system is captured by an infinite tree: the unraveling of the transition graph.
- Properties of the system behavior can be specified as properties of infinite trees.

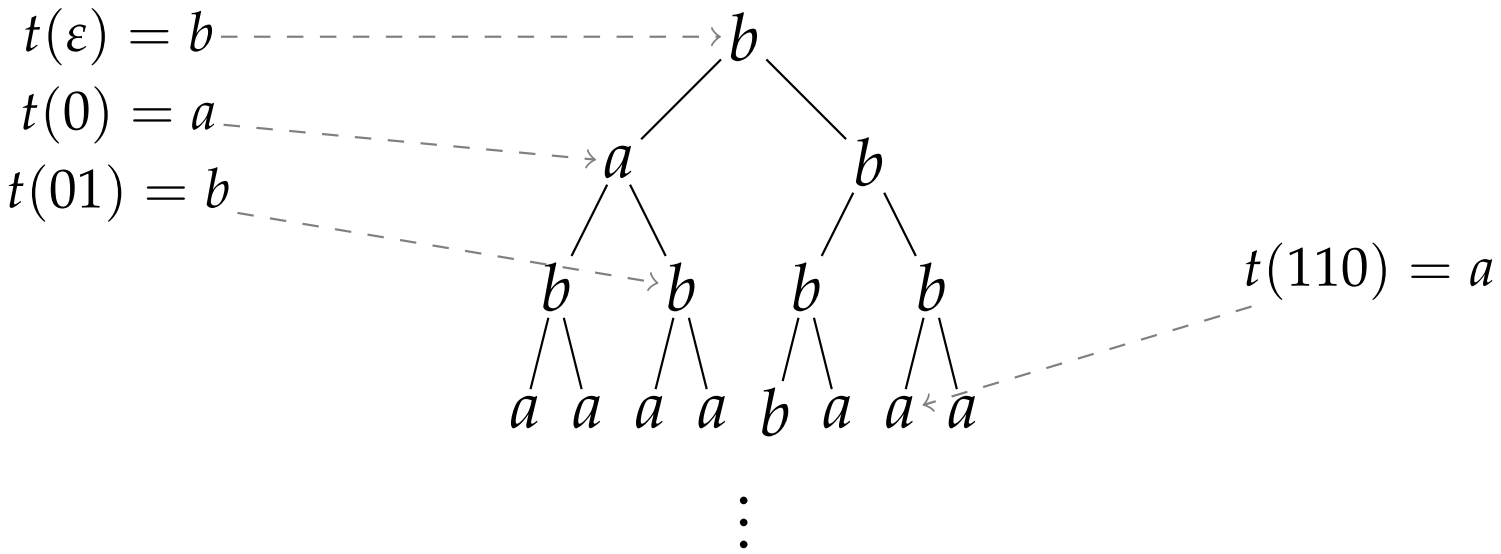
Infinite trees

- For simplicity we restrict ourselves to complete binary trees.
- The nodes are labeled from a finite alphabet Σ.
- Formally, a tree is a mapping $t : \{0,1\}^* \to \Sigma$.

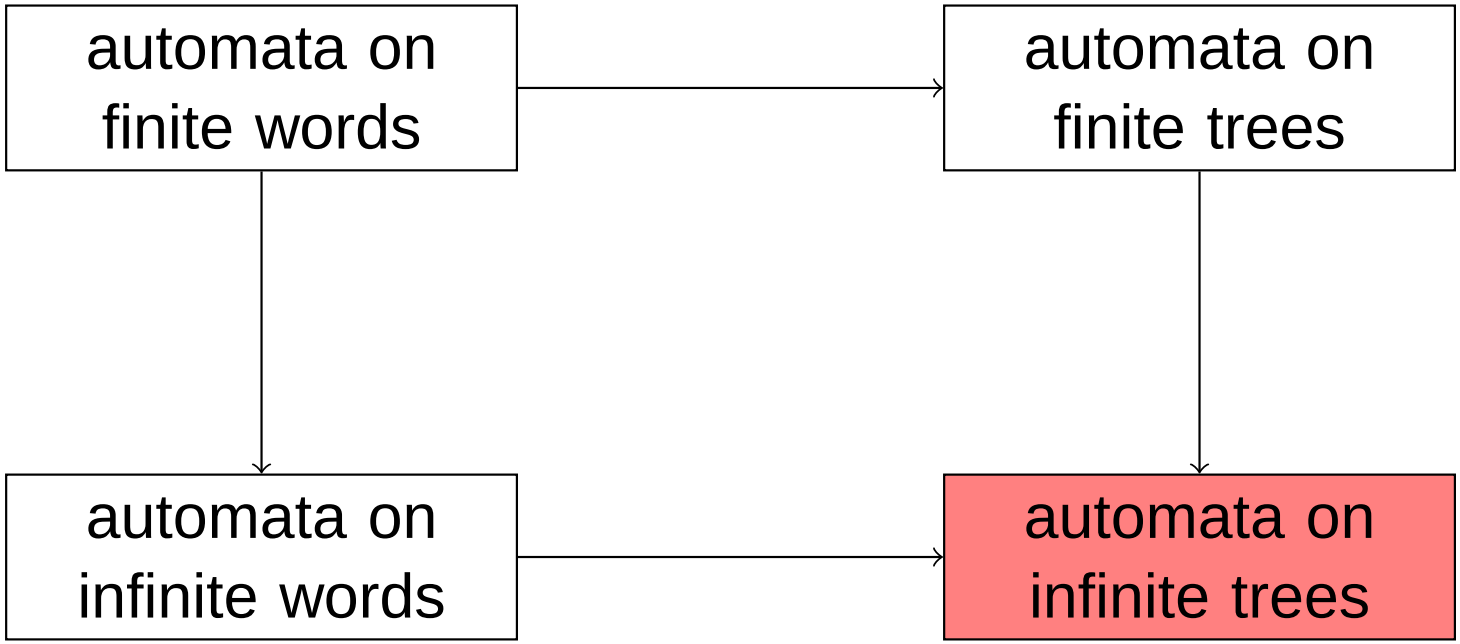


Infinite trees

- For simplicity we restrict ourselves to complete binary trees.
- The nodes are labeled from a finite alphabet Σ.
- Formally, a tree is a mapping $t : \{0,1\}^* \to \Sigma$.



Automata on infinite trees

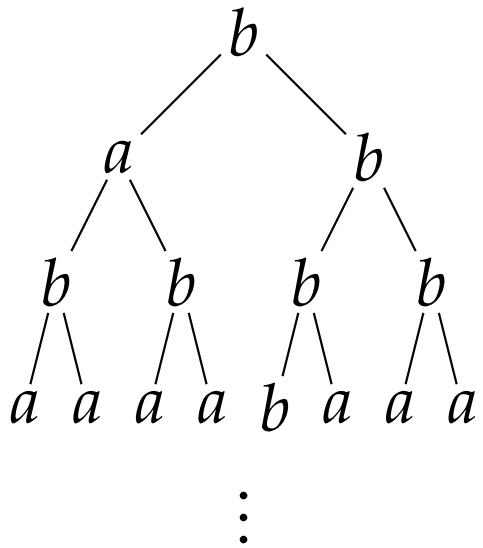


- Robust model (good closure and algorithmic properties)
- Captures many known specification logics

Parity tree automata

- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')

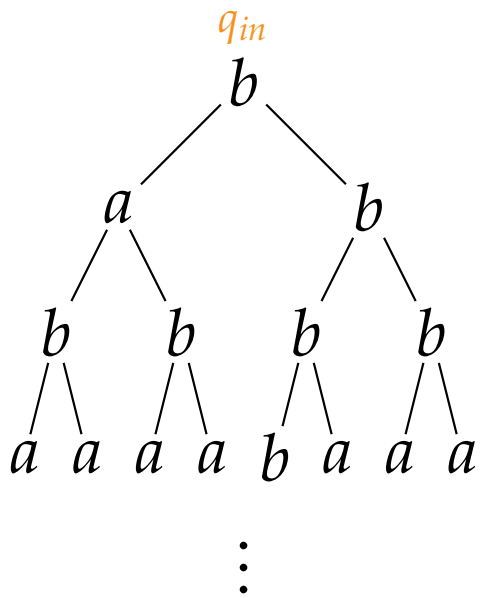
Run of $\mathcal{A}$:



Parity tree automata

- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')

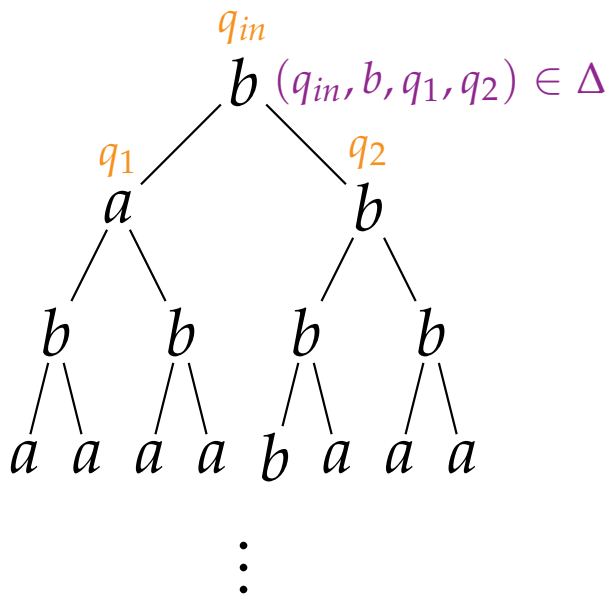
Run of $\mathcal{A}$:



Parity tree automata

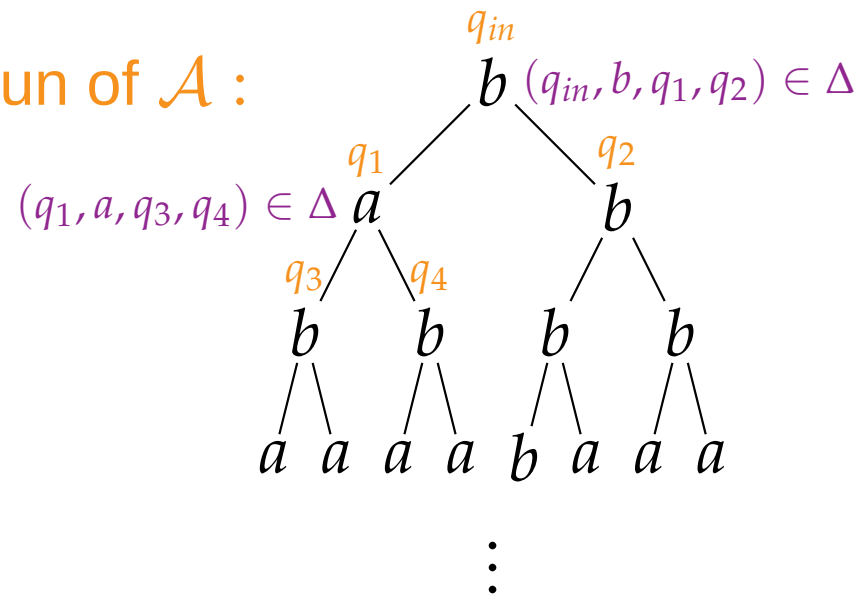
- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')

Run of $\mathcal{A}$:



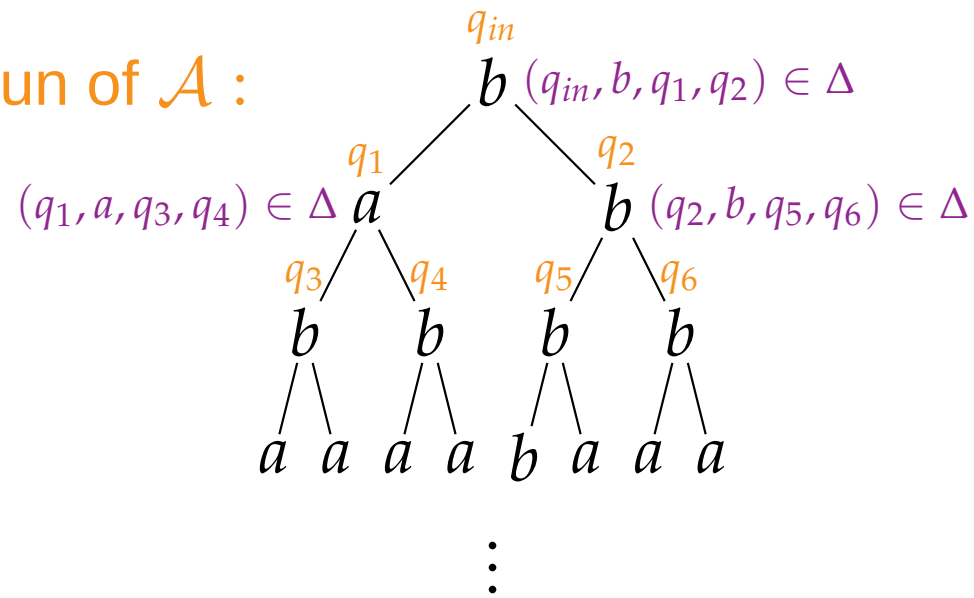
Parity tree automata

- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')



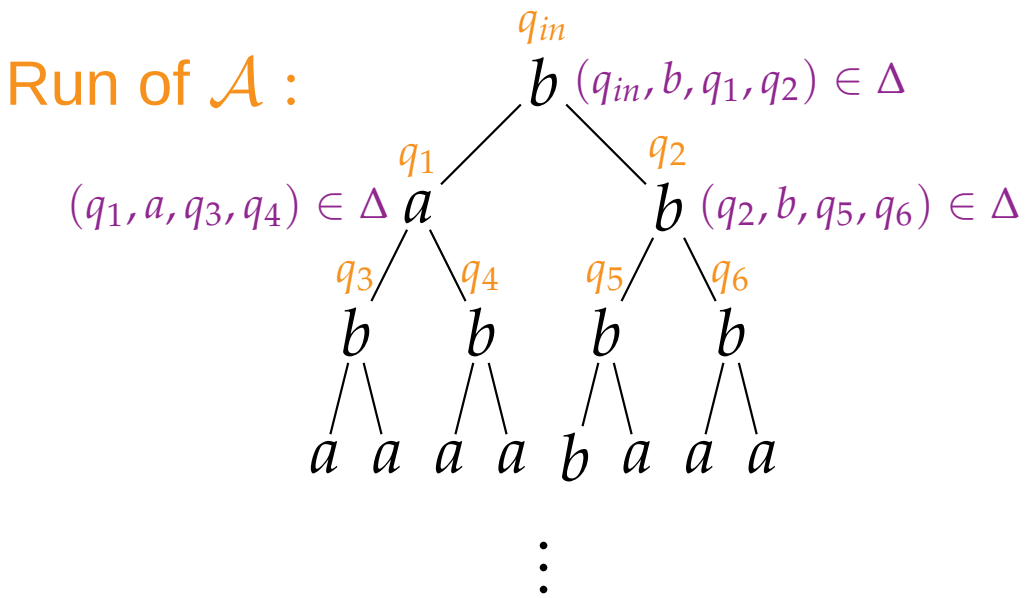
Parity tree automata

- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')



Parity tree automata

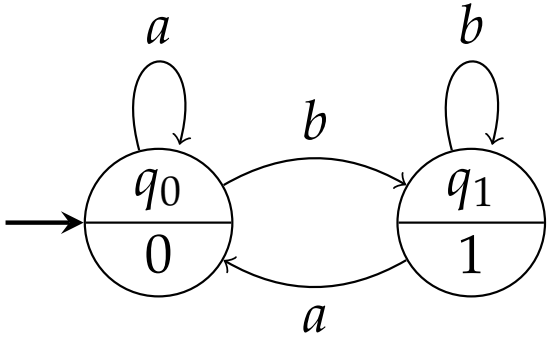
- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')



- Priority function $pri : Q \to \mathbb{N}$
- Run accepting if on each path the maximal priority appearing infinitely often is even.
- Tree accepted if there is an accepting run on this tree. $T(\mathcal{A})$ denotes the language of accepted trees.

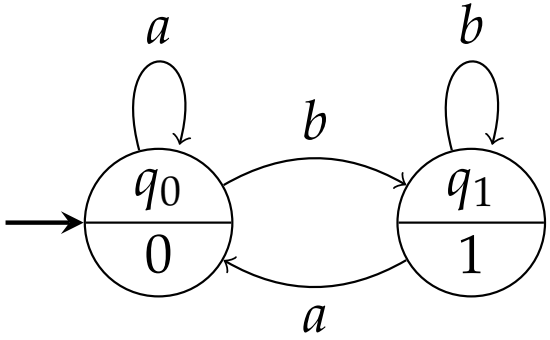
Example

- Trees over $\{a, b\}$ such that on each infinite path there are only finitely many b
- Use parity word automaton and run it on all branches



Example

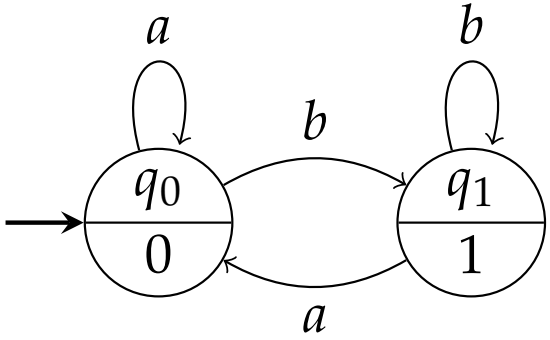
- Trees over $\{a, b\}$ such that on each infinite path there are only finitely many b
- Use parity word automaton and run it on all branches



- Required transitions: (q_0, a, q_0, q_0), (q_0, b, q_1, q_1), (q_1, a, q_0, q_0), and (q_1, b, q_1, q_1)

Example

- Trees over $\{a, b\}$ such that on each infinite path there are only finitely many b
- Use parity word automaton and run it on all branches



- Required transitions: (q_0, a, q_0, q_0), (q_0, b, q_1, q_1), (q_1, a, q_0, q_0), and (q_1, b, q_1, q_1)
- There is no (nondeterministic) Büchi tree automaton for this language.

Example

The set of all trees t over $\{a, b\}$ such that t contains at least one node labeled b:

(q_b, a, q_b, q) (q_b, a, q, q_b) (q_b, b, q, q)

$pri(q_b) = 1$ and $pri(q) = 0$

Nondeterminsm is required

Acceptance conditions

- As for games we can define Muller, Rabin, Streett, Büchi, etc. automata.
- The LAR construction can be used to transform Muller tree automata into parity tree automata.
- Call a language of infinite trees regular if it can be accepted by a parity tree automaton.

Basic closure properties

Proposition

The class of regular languages of infinite trees is closed under union, intersection, and projection (relabeling).

Basic closure properties

Proposition

The class of regular languages of infinite trees is closed under union, intersection, and projection (relabeling).

Proof:

- Union, intersection: A standard product construction yields a Muller automaton over pairs of priorities.
- Projection $h : \Sigma \to \Gamma$: simply apply the projection to the labels in the transitions.

Basic closure properties

Proposition

The class of regular languages of infinite trees is closed under union, intersection, and projection (relabeling).

Proof:

- Union, intersection: A standard product construction yields a Muller automaton over pairs of priorities.
- Projection $h : \Sigma \to \Gamma$: simply apply the projection to the labels in the transitions.

What about complementation?

Complementation

Complementation

Tree accepted:

$$\exists \text{run} \forall \text{path}.(\text{path satisfies acceptance condition})$$

Tree not accepted:

$$\forall \text{run} \exists \text{path}.(\text{path does not satisfy acceptance condition})$$

This exchange of quantifiers makes the problem difficult.

Complementation

Tree accepted:

$$\exists \text{run} \forall \text{path}.(\text{path satisfies acceptance condition})$$

Tree not accepted:

$$\forall \text{run} \exists \text{path}.(\text{path does not satisfy acceptance condition})$$

This exchange of quantifiers makes the problem difficult.

Idea for solution

Reformulate acceptance as:

$$\exists \text{strategy for Eva } \forall \text{strategies for Adam } (...)$$

Complementation

Tree accepted:

$$\exists \text{run} \forall \text{path}.(\text{path satisfies acceptance condition})$$

Tree not accepted:

$$\forall \text{run} \exists \text{path}.(\text{path does not satisfy acceptance condition})$$

This exchange of quantifiers makes the problem difficult.

Idea for solution

Reformulate acceptance as:

$$\exists \text{strategy for Eva } \forall \text{strategies for Adam } (...)$$

Determinacy yields for non-acceptance:

$$\exists \text{strategy for Adam } \forall \text{strategies for Eva } (...)$$

Membership game

Use a game to characterize when a tree t is accepted by $\mathcal{A}$:

- Player Constructor tries to show that t is accepted by constructing a run.
- Player Spoiler tries to show the contrary by selecting a path on which the acceptance condition is not satisfied.

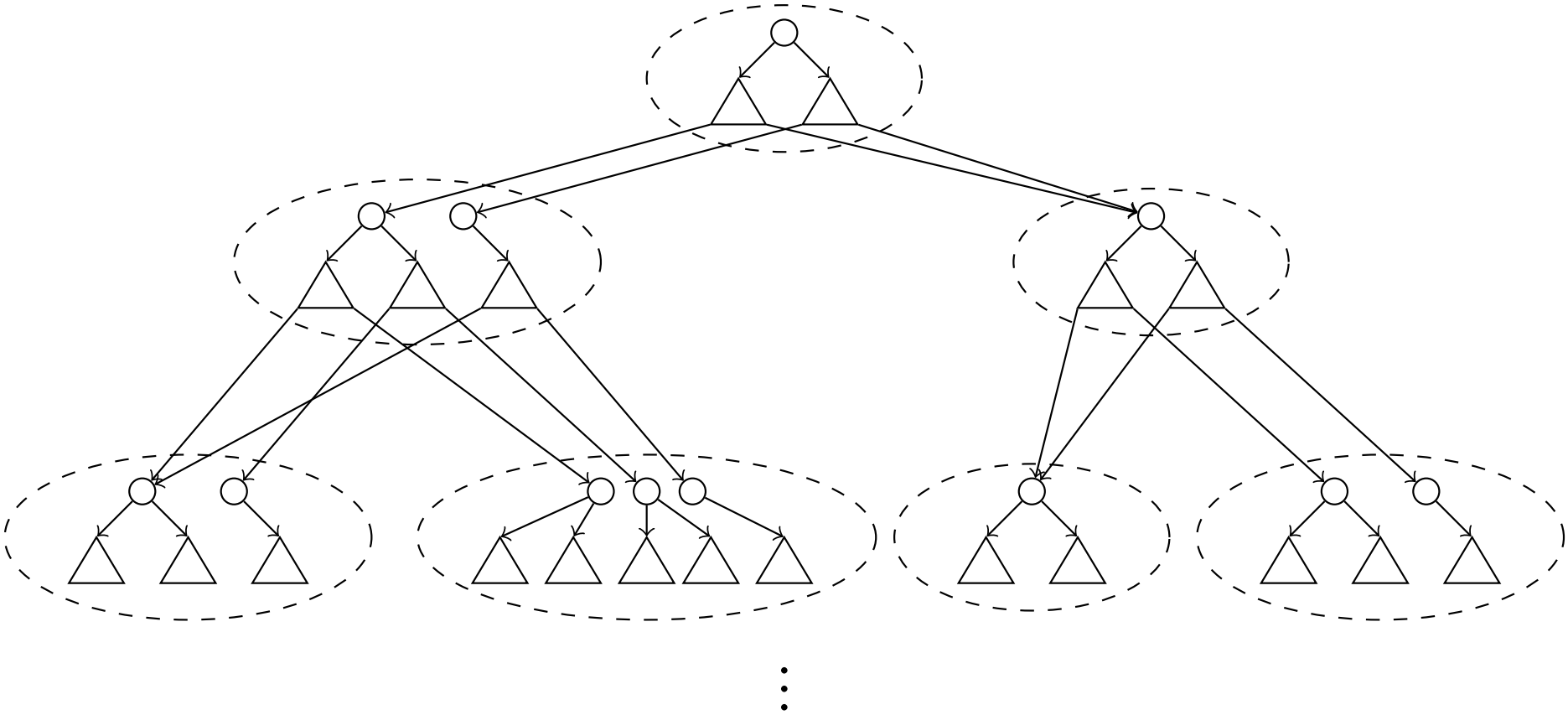
Membership game

Use a game to characterize when a tree t is accepted by $\mathcal{A}$:

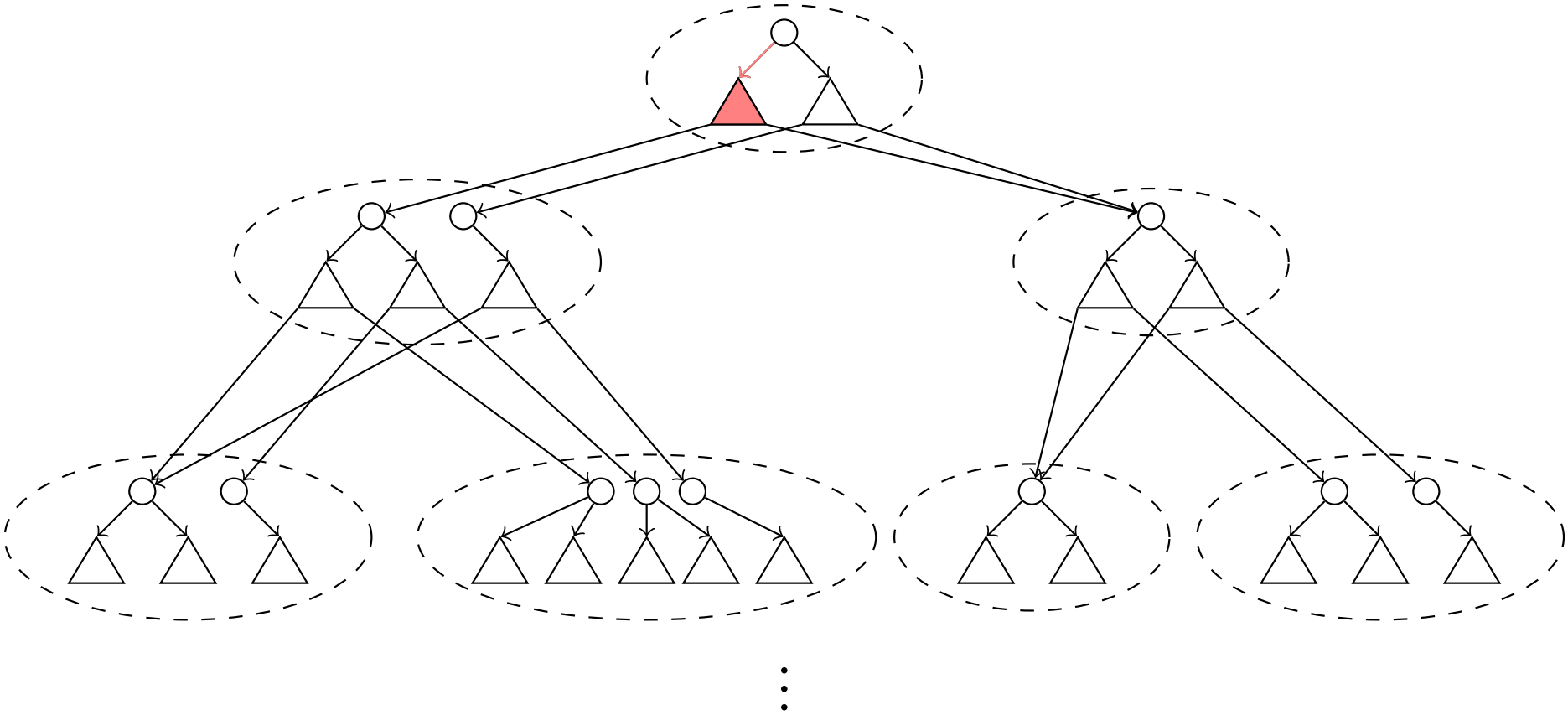
- Player Constructor tries to show that t is accepted by constructing a run.
- Player Spoiler tries to show the contrary by selecting a path on which the acceptance condition is not satisfied.
- The game starts a the root of the tree in the initial state of $\mathcal{A}$.
- The moves of the game from a position (u, q) where u is a node of t, and q is a state of $\mathcal{A}$:
  1. Constructor picks a transition (q, a, q_0, q_1) that matches q and the label a of t at u
  2. Spoiler chooses a direction and the game moves on to position $(u0, q_0)$ or $(u1, q_1)$.
- Winning condition for Constructor: acceptance condition of $\mathcal{A}$

Parity game on an infinite graph

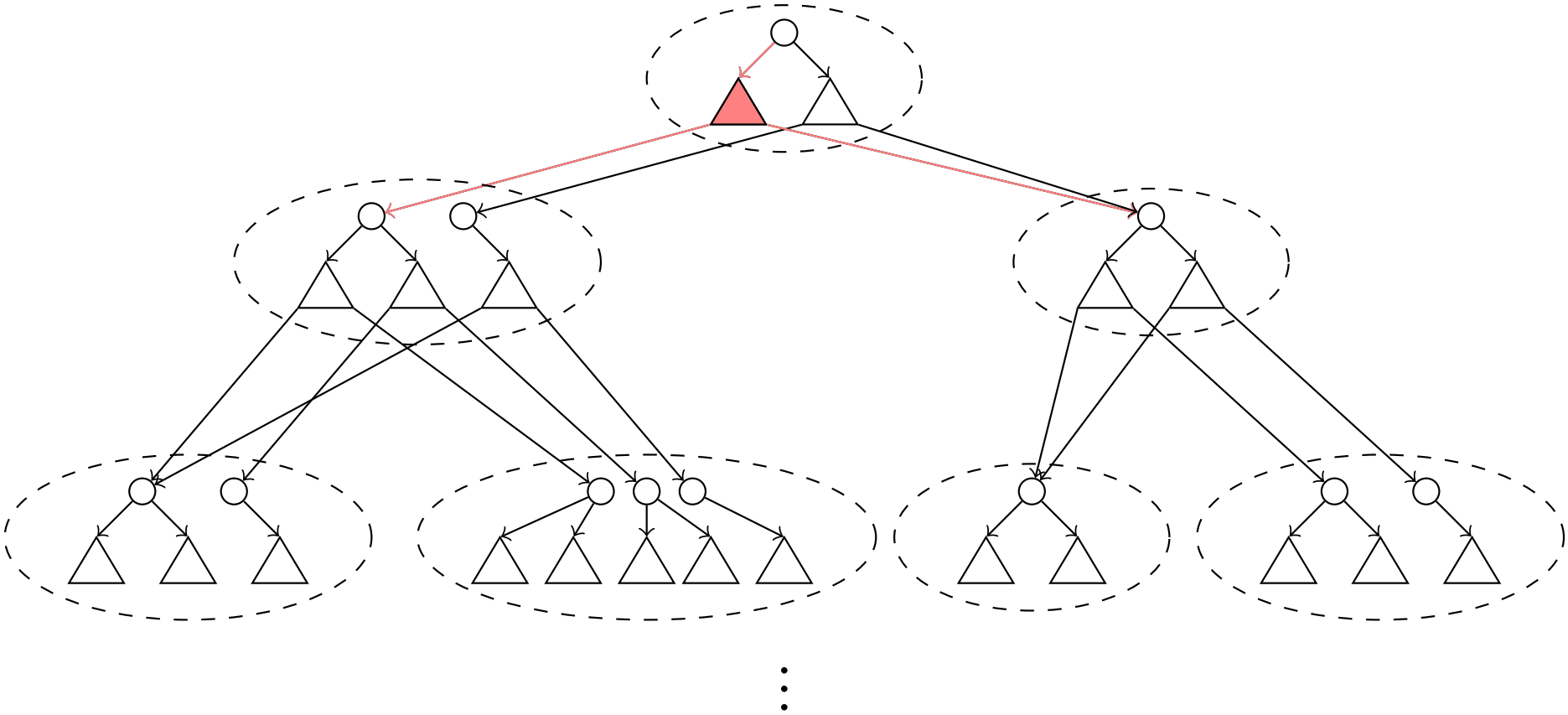
Shape of the membership game



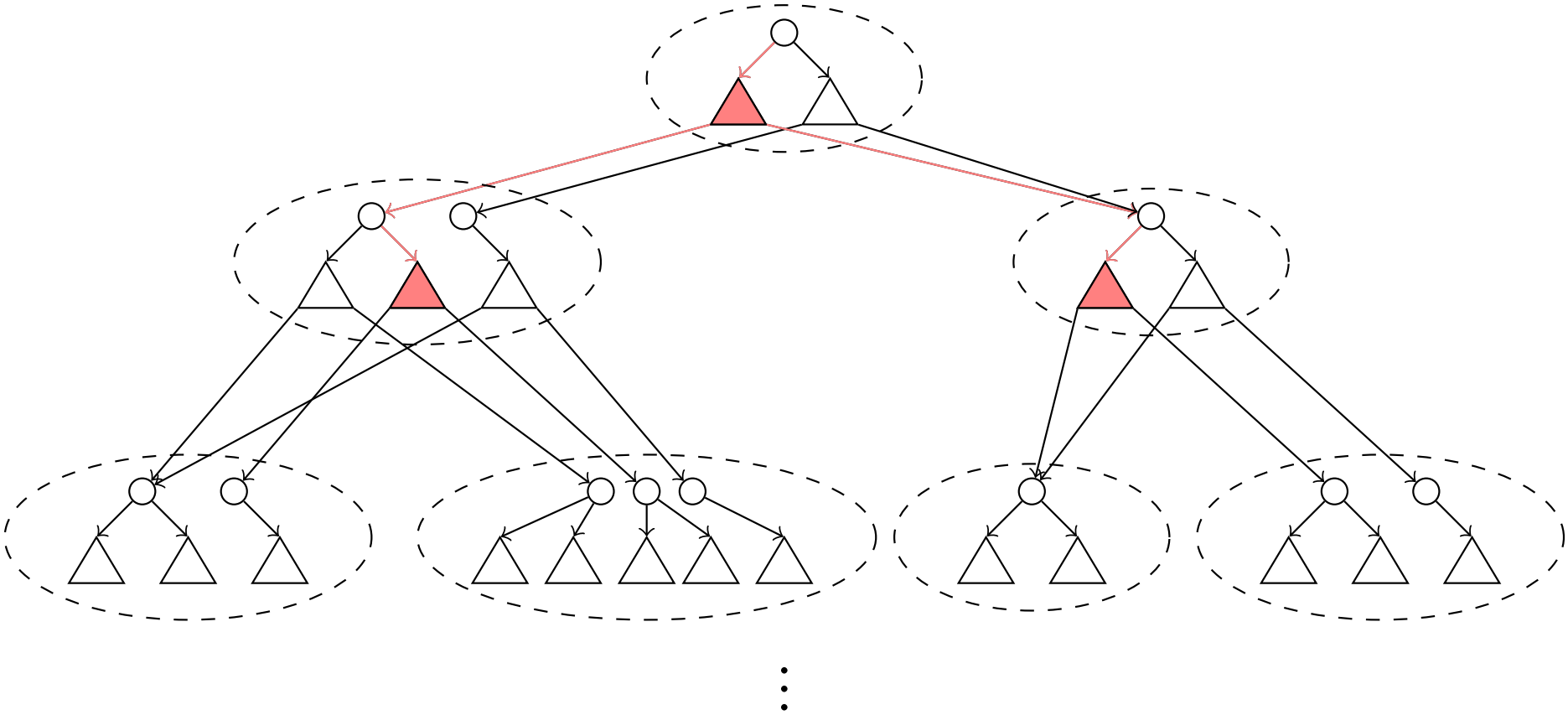
Shape of the membership game



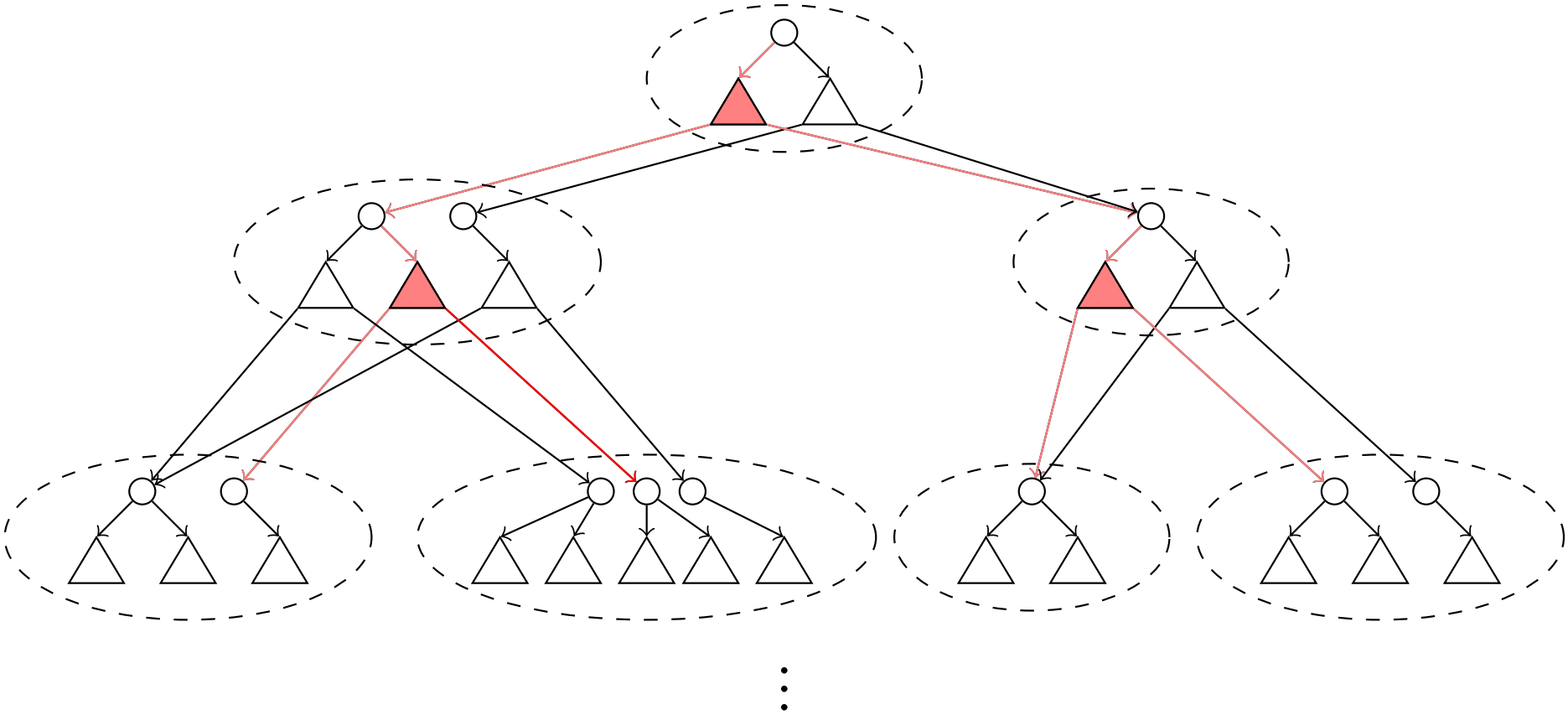
Shape of the membership game



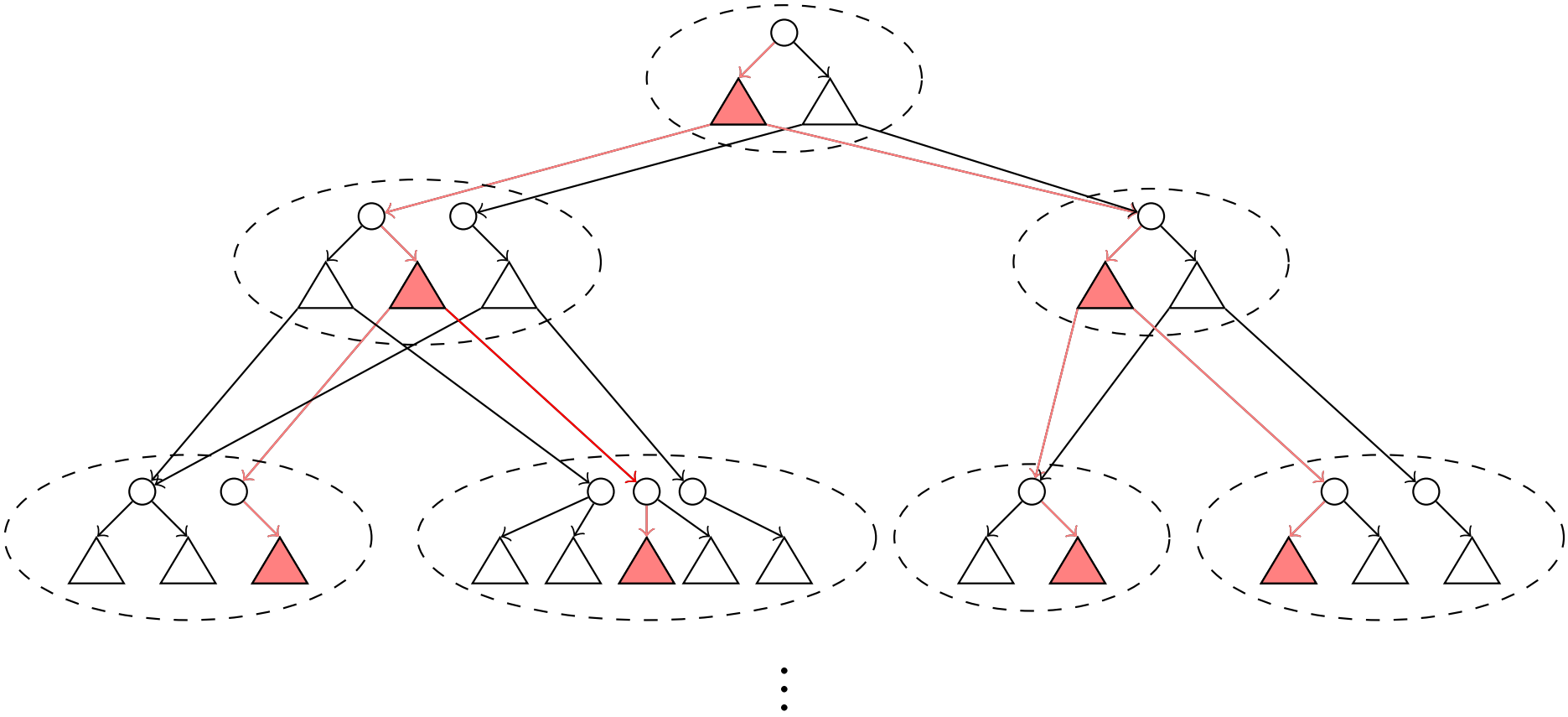
Shape of the membership game



Shape of the membership game

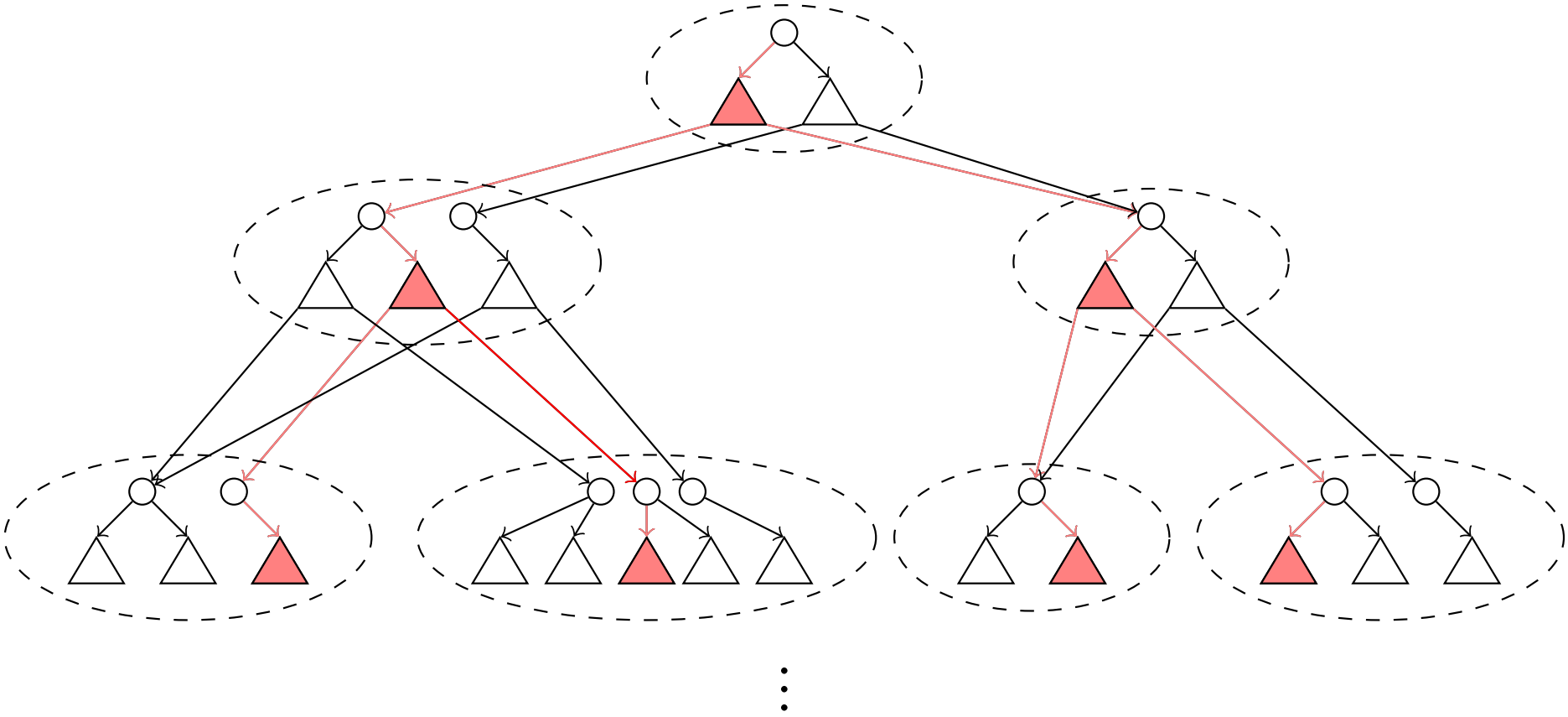


Shape of the membership game



winning strategy of CONSTRUCTOR $\rightsquigarrow$ accepting run

Shape of the membership game



winning strategy of CONSTRUCTOR $\rightsquigarrow$ accepting run

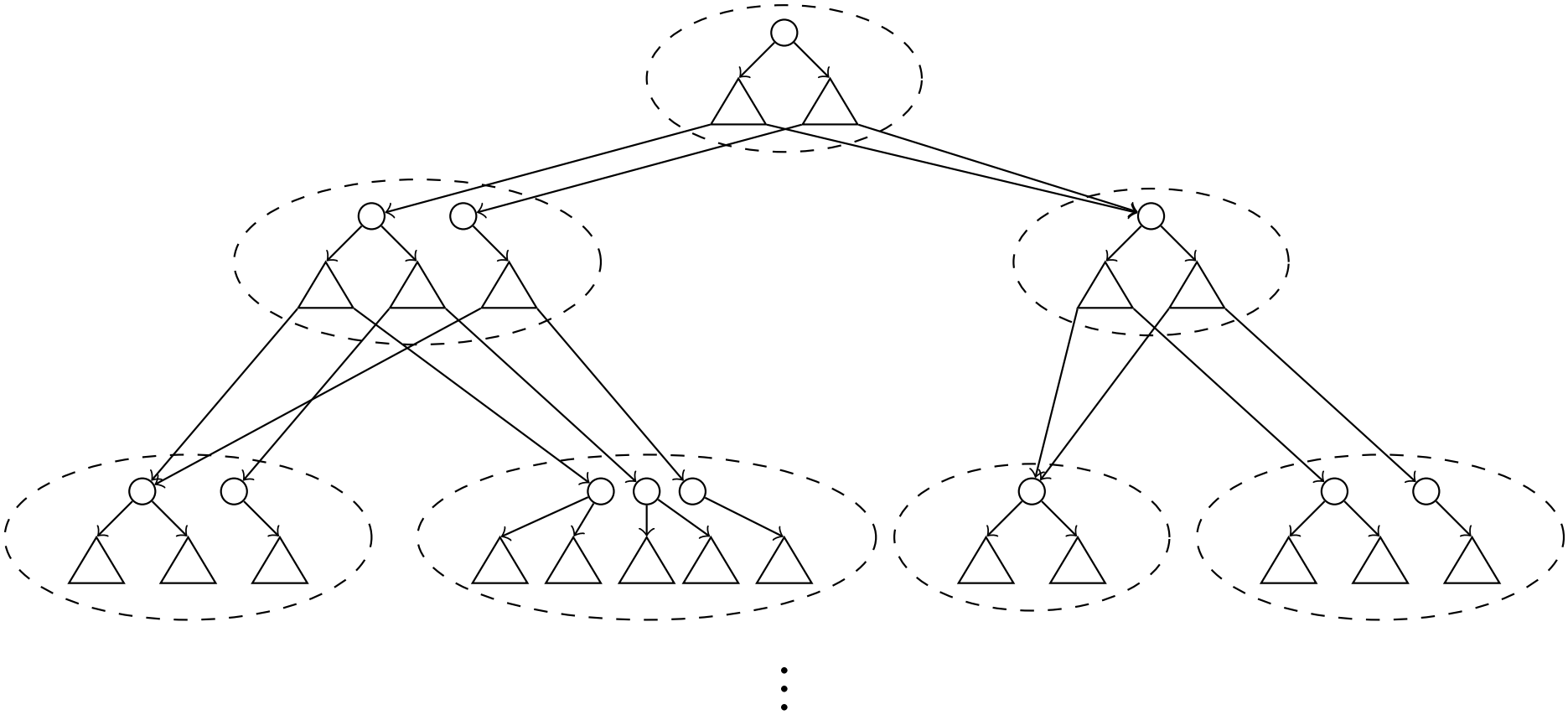
accepting run $\rightsquigarrow$ winning strategy of CONSTRUCTOR

Applying positional determinacy

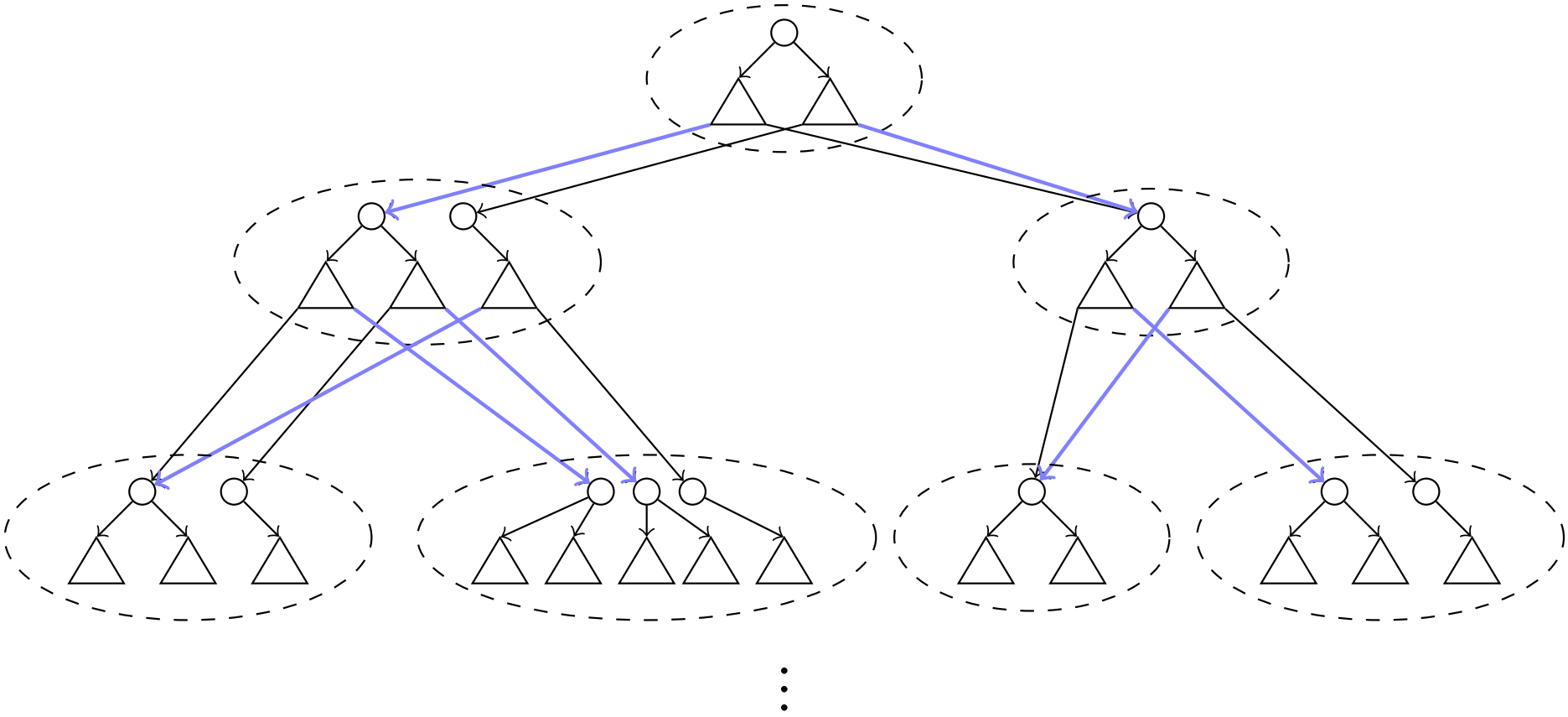
Lemma

A tree t is not in $T(\mathcal{A})$ iff
there is a positional winning strategy for SPOILER
in the membership game.

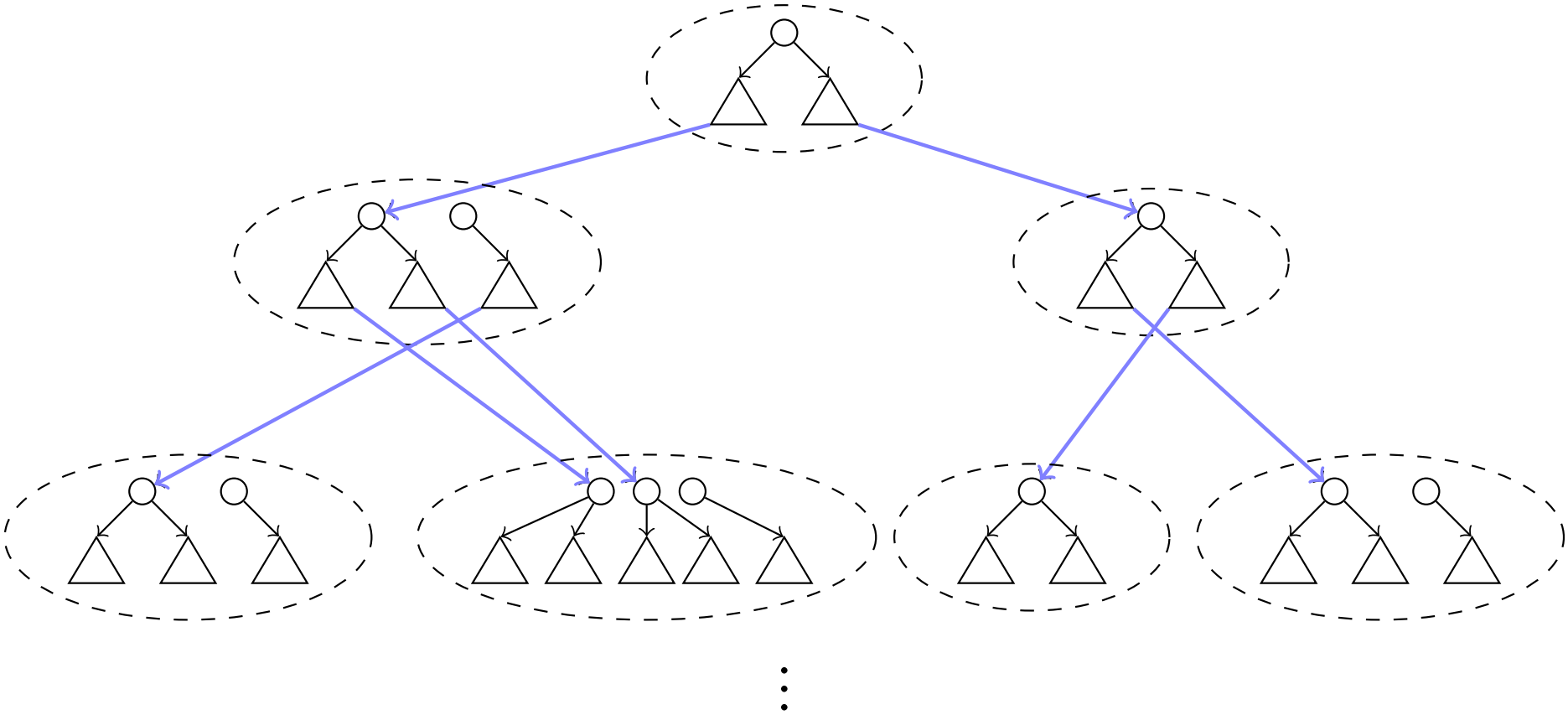
Strategies for SPOILER



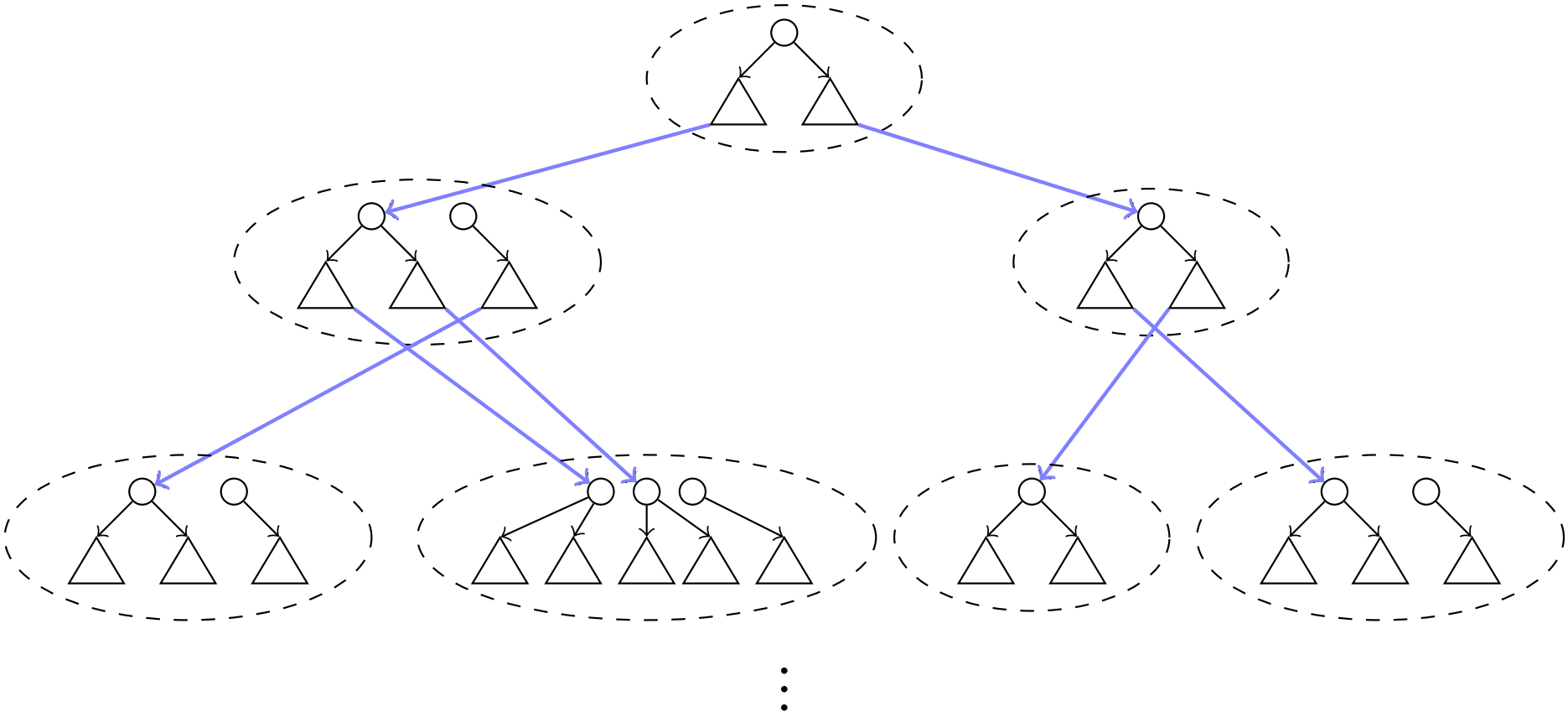
Strategies for SPOILER



Strategies for SPOILER



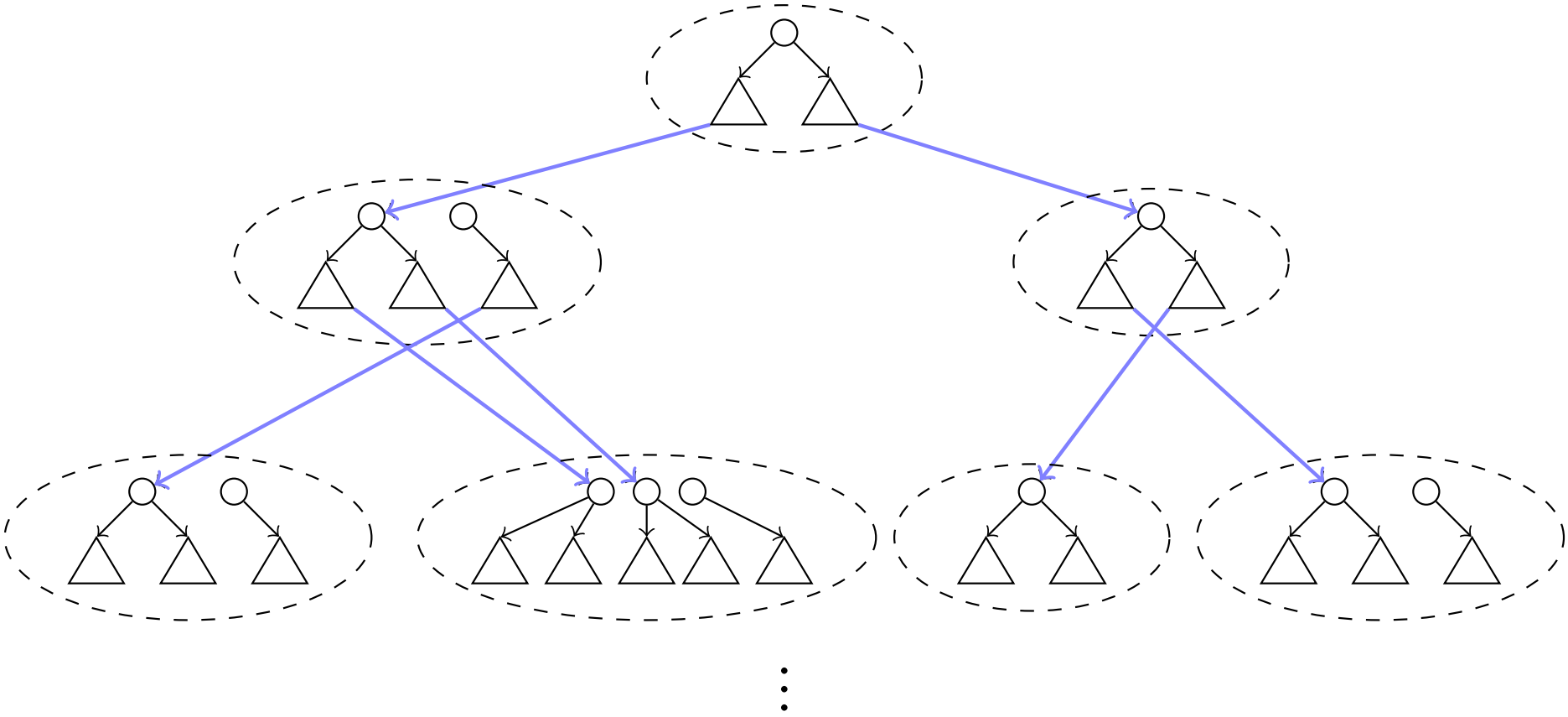
Strategies for Spoiler



Postional strategy for Spoiler coded as

$$\sigma : \{0,1\}^* \to \underbrace{(\Delta \to \{0,1\})}_{\Gamma}$$

Strategies for Spoiler



Postional strategy for Spoiler coded as

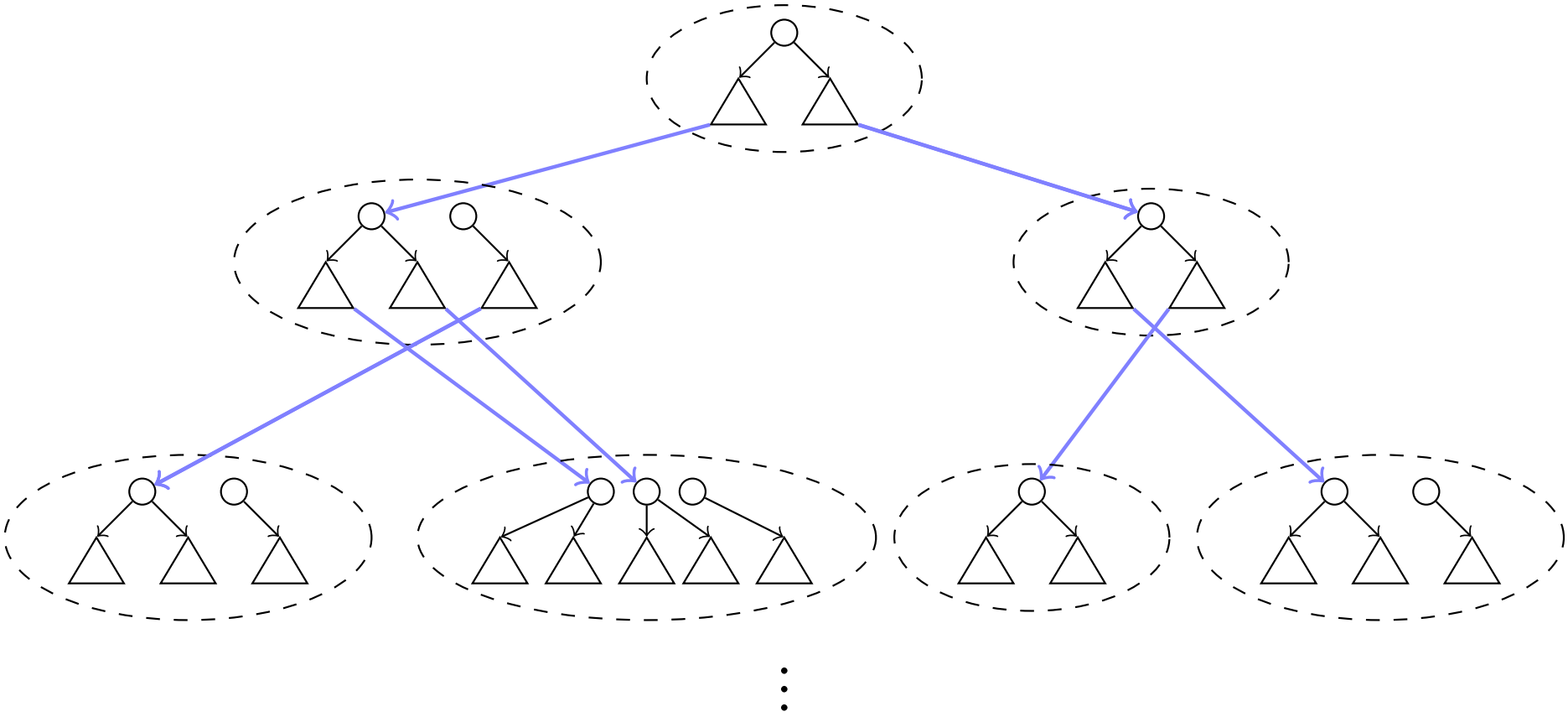
$$\sigma : \{0,1\}^* \to \underbrace{(\Delta \to \{0,1\})}_{\Gamma}$$

This is a tree over the alphabet Γ

Next steps

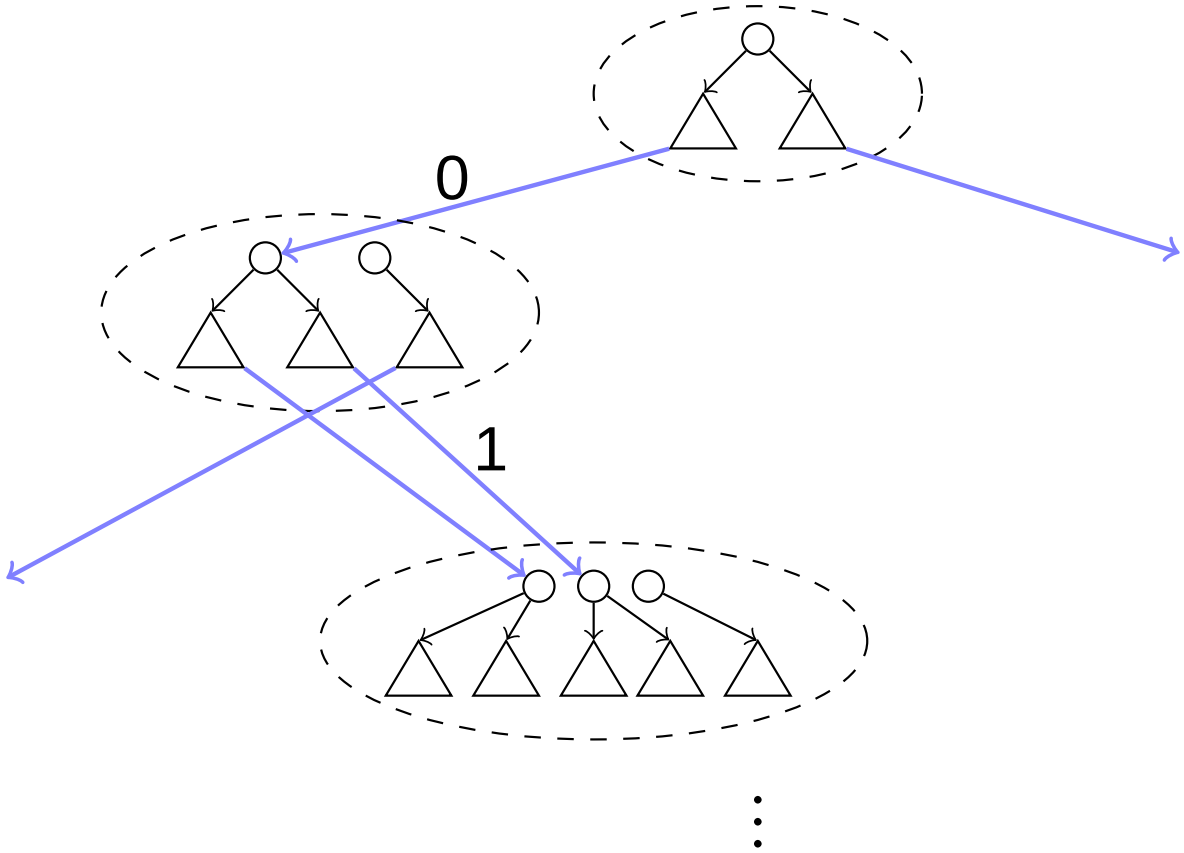
- Construct an automaton $\mathcal{A}_{\text{strat}}$ that reads trees of the form $t \times \sigma$, i.e., annotated with a positional strategy for Spoiler
- $\mathcal{A}_{\text{strat}}$ accepts $t \times \sigma$ if σ is winning for Spoiler
- Obtain $\mathcal{C}$ from $\mathcal{A}_{\text{strat}}$ by omitting the strategy annotation in the labels ($\mathcal{C}$ guesses the strategy for Spoiler).

Construction of $\mathcal{A}_{\text{strat}}$



$\mathcal{A}_{\text{strat}}$ has to check: the paths following the blue edges do not satisfy the acceptance condition

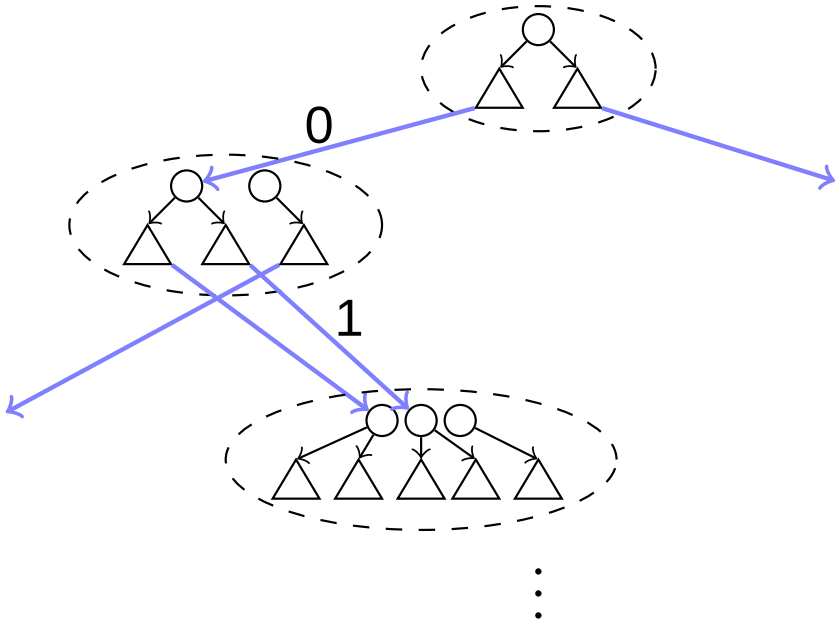
Construction of $\mathcal{A}_{\text{strat}}$



$\mathcal{A}_{\text{strat}}$ has to check: the paths following the blue edges do not satisfy the acceptance condition

Focus on single branches: infinite words over $\Sigma \times \Gamma \times \{0, 1\}$

Verifying single branches



- Construct a nondeterministic Büchi automaton that guesses a blue path and accepts if it **does** satisfy the acceptance condition of $\mathcal{A}$.
- Determinize and complement, and obtain a deterministic parity word automaton $\mathcal{A}^{\text{path}}_{\text{strat}}$ that accepts those branches of a tree on which Spoiler's strategy is winning.

Back to trees

- Run $\mathcal{A}^{\text{path}}_{\text{strat}}$ along each branch:

$$\mathcal{A}^{\text{path}}_{\text{strat}}: \quad \delta(q,(a,\gamma,0)) = q' \qquad \delta(q,(a,\gamma,1)) = q''$$

$$\mathcal{A}_{\text{strat}}: \qquad\qquad (q,(a,\gamma),q',q'')$$

- Then obtain $\mathcal{C}$ by omitting the strategy encoding:

$(q,(a,\gamma),q',q'')$ becomes (q,a,q',q'')

Summary of the complementation method

- Characterize acceptance in terms of winning strategy in membership game
- Positional determinacy for parity games yields: t not accepted iff SPOILER has positional winning strategy
- Construct an automaton that checks if a given strategy of SPOILER is winning.
  - This construction is based on the determinization of ω-automata.
- Obtain the desired automaton by projection (removing the strategy annotations).

Closure under complementation

Theorem (Rabin'69)

For a given tree automaton one can construct a tree automaton for the complement language.

From logic to automata

- Closure properties allow translation of S2S into tree automata
- S2S is the monadic second-order logic over the infinite binary tree, i.e., the structure $(\{0,1\}^*, S_0, S_1)$ consisting of
  - the tree nodes as domain
  - the two successor functions
- The translation works in the same way as for S1S to ω-automata
- The satisfiability problem for S2S becomes the emptiness problem for tree automata

Emptiness

Emptiness problem

Decide for a given PTA $\mathcal{A}$ if $T(\mathcal{A})$ is empty.

Idea for solution:

- In the membership game for t, Constructor's task is to construct an accepting run on t
- In the emptiness game, Constructor's task is to construct a tree along with an accepting run on t

From membership to emptiness

How to adapt the membership game to obtain the emptiness game:

- The game starts at the root of the tree t in the initial state of $\mathcal{A}$.
- The moves of the game from a position (u, q) where u is a node of t, and q is a state of $\mathcal{A}$:
  1. Constructor picks a transition (q, a, q_0, q_1) that matches q and the label a of t at u
  2. Spoiler chooses a direction and the game moves on to position $(u0, q_0)$ or $(u1, q_1)$.
- Winning condition for Constructor: acceptance condition of $\mathcal{A}$

From membership to emptiness

How to adapt the membership game to obtain the emptiness game:

- The game starts at the root of the tree t in the initial state of $\mathcal{A}$.
- The moves of the game from a position (u, q) where u is a node of t, and q is a state of $\mathcal{A}$:
  1. Constructor picks a transition (q, a, q_0, q_1) that matches q ~~and the label a of t at u~~
  2. Spoiler chooses a direction and the game moves on to position $(u0, q_0)$ or $(u1, q_1)$.
- Winning condition for Constructor: acceptance condition of $\mathcal{A}$

From membership to emptiness

How to adapt the membership game to obtain the emptiness game:

- The game starts ~~at the root of the tree t~~ in the initial state of $\mathcal{A}$.
- The moves of the game from a position $(~~u~~, q)$ where ~~u is a node of t~~, and q is a state of $\mathcal{A}$:
  1. Constructor picks a transition (q, a, q_0, q_1) that matches q ~~and the label a of t at u~~
  2. Spoiler chooses a direction and the game moves on to position $(~~u0~~, q_0)$ or $(~~u1~~, q_1)$.
- Winning condition for Constructor: acceptance condition of $\mathcal{A}$

Emptiness game

The emptiness game for $\mathcal{A}$:

- The game positions are $Q \cup \Delta$ (states and transitions of $\mathcal{A}$)
- The moves of the game from a position q (a state of $\mathcal{A}$):
  1. Constructor picks a transition (q, a, q_0, q_1)
  2. Spoiler chooses a direction and the game moves on to position q_0 or q_1
- Winning condition for Constructor: acceptance condition of $\mathcal{A}$

This is a finite game graph.

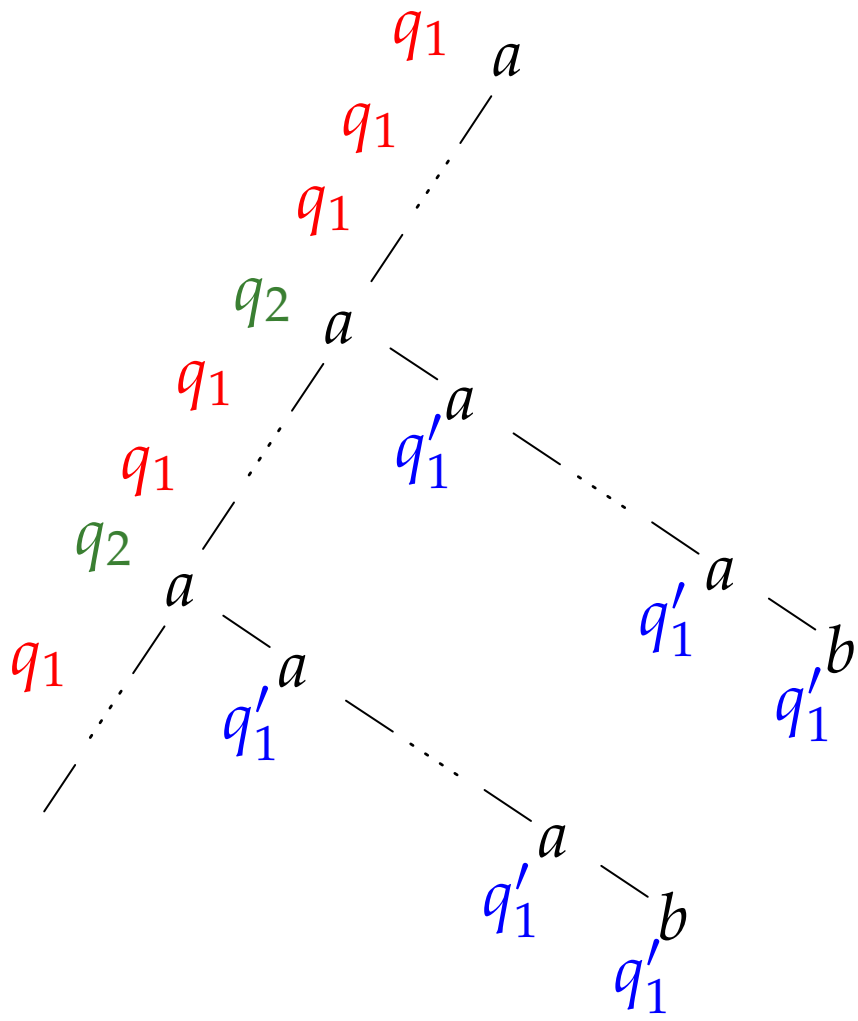
Lemma

Constructor *has a winning strategy in the emptiness game for* $\mathcal{A}$ *iff* $T(\mathcal{A}) \neq \emptyset$

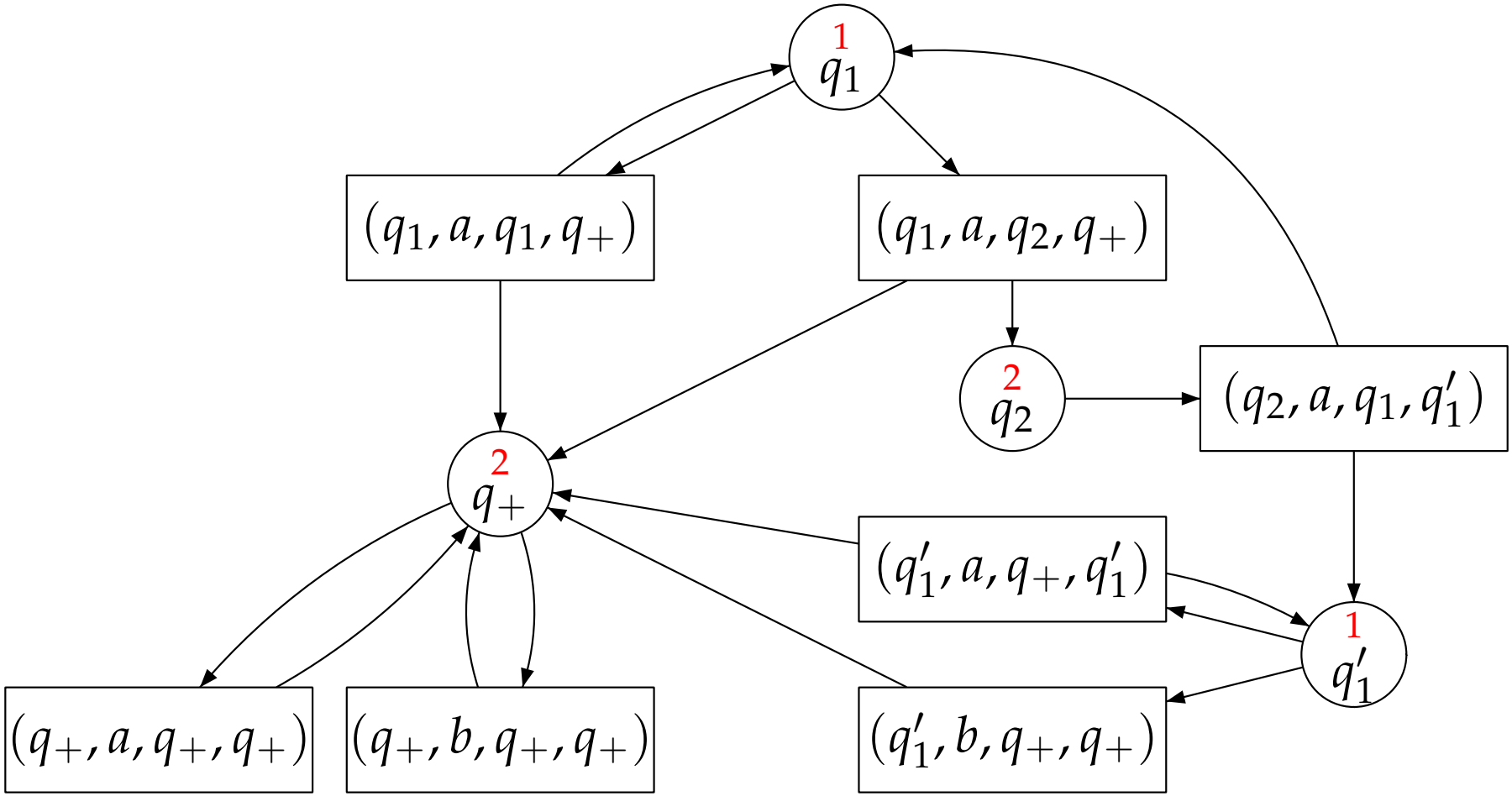
Example automaton

(q_1, a, q_1, q_+)
(q_1, a, q_2, q_+)
(q_2, a, q_1, q_1')
(q_1', a, q_+, q_1')
(q_1', b, q_+, q_+)
(q_+, a, q_+, q_+)
(q_+, b, q_+, q_+)

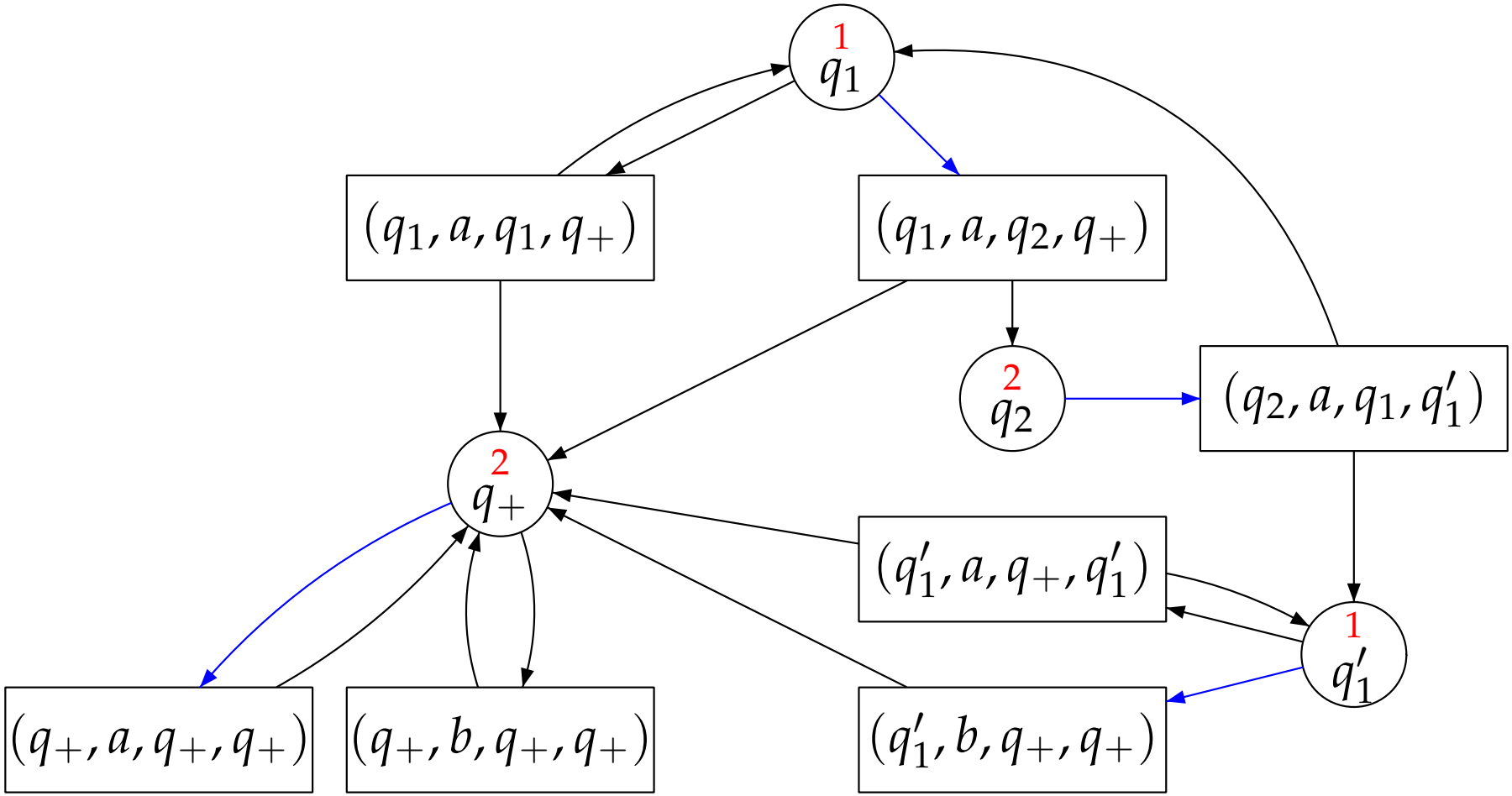
$c(q_1) = c(q_1') = 1$
$c(q_2) = c(q_+) = 2$



Emptiness game



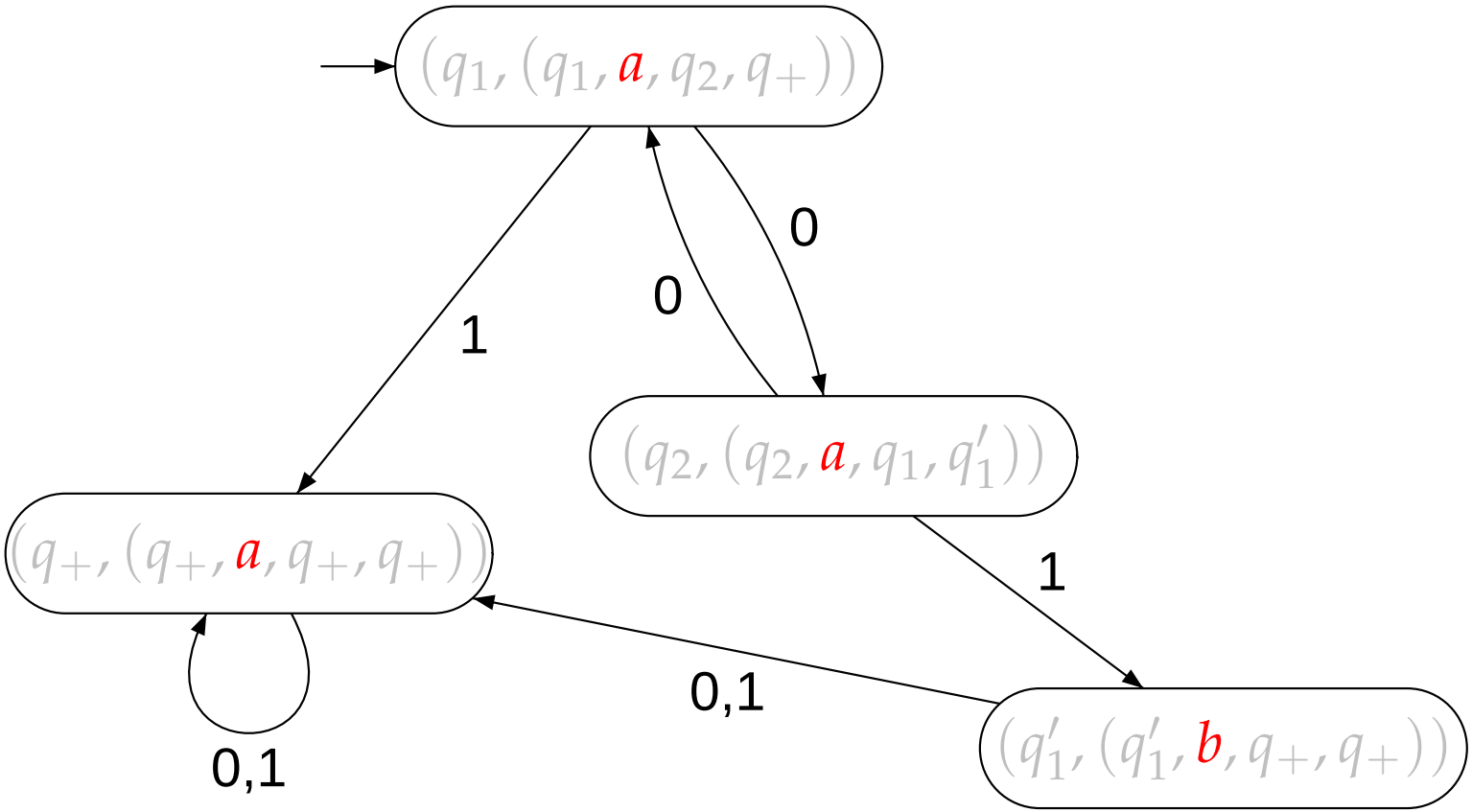
Emptiness game



A winning strategy for CONSTRUCTOR

Regular trees

From the winning strategy: a finitely generated (regular) tree in the language



Emptiness decidable

Theorem (Rabin'69)

The emptiness problem for parity tree automata is decidable (in NP ∩ co-NP). If the language is not empty, then one can construct a finite representation of a tree in the language.

Corollary (Rabin'69)

S2S is decidable.

Summary of this section

Games as a tool for tree automata constructions:

- For complementation of parity tree automata the **positional determinacy of parity games is used for the correctness proof** of the construction.
  The actual construction is based on determinization of word automata.
- The **emptiness test directly reduces to a game**. A positional winning strategy for CONSTRUCTOR yields a regular tree in the language.

1 Introduction

2 Basics on games

3 Transformation of winning conditions

- ω-Automata
- Game reductions
- Logical winning conditions

4 Tree automata

- Complementation
- Emptiness

5 Beyond finite automata

Context-free specifications

What happens if specifications are not ω-regular?

Proposition

The problem of finding the winner in a game (G, Win), where Win is defined by a nondeterministic pushdown ω-automaton is undecidable.

Pushdown games

Assume Win is specified by a deterministic pushdown ω-automaton (with parity condition):

Taking the product of G with the pushdown automaton results in a parity game on a pushdown graph. These games can be solved and winning strategies can be implemented by pushdown automata.

Theorem (Walukiewicz'96)

Parity games on pushdown graphs can be solved in exponential time and winning strategies can be implemented by pushdown automata.

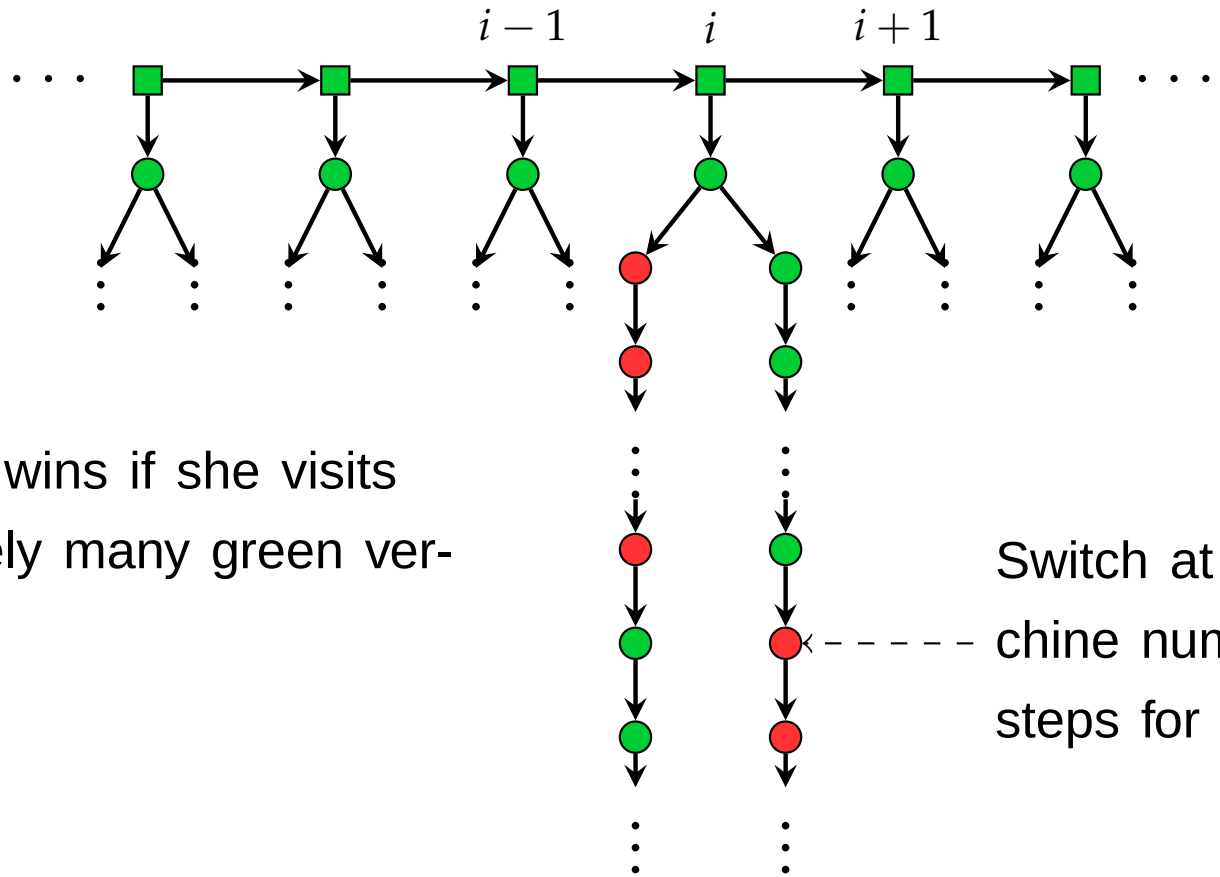
Corollary

The problem of deciding the winner in a game (G, Win) for a finite game graph, where Win is defined by a deterministic pushdown ω-automaton (with parity condition) is decidable.

Recursive games

In general, one can study games on recursive game graphs (infinite set of vertices, computable edge relation and coloring).

For recursive parity games winning strategies need not be computable:



Eva wins if she visits finitely many green vertices

Switch at level k if Turing machine number i halts after k steps for the empty input

Summary

1. Solving infinite games with the help of automata: Extend the game by running a **deterministic** automaton along the plays
   - Determinization: nondeterministic Büchi $\longrightarrow$ deterministic parity
   - Muller conditions: LAR construction
   - LTL conditions: nondeterministic automaton guessing valuations of subformulas
   - S1S conditions: inductive translation to automata
   - Solution to Church's synthesis problem
2. Games as a tool for tree automata constructions
   - Complementation
   - Emptiness test
   - Application: decidability of S2S
3. Nonregular conditions and infinite graphs
   - Pushdown games
   - Recursive games