Games with Structured States

Łukasz Kaiser

Mathematische Grundlagen der Informatik
RWTH Aachen

LIAFA Seminar

Paris, 2010

Overview

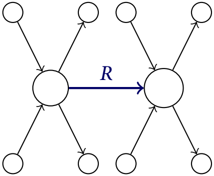
Structure Rewriting

Separated Games

Simulation-Based Playing

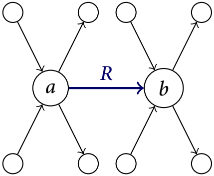
Structure Rewriting Rules

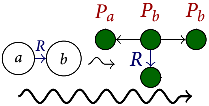
Rewriting Example



Structure Rewriting Rules

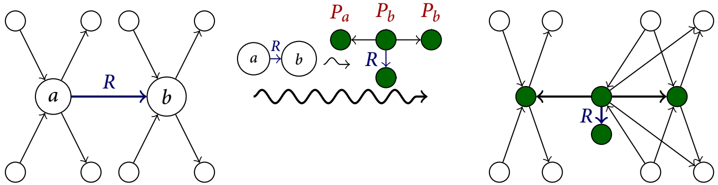
Rewriting Example





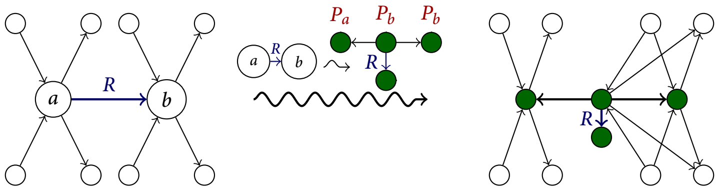
Structure Rewriting Rules

Rewriting Example



Structure Rewriting Rules

Rewriting Example



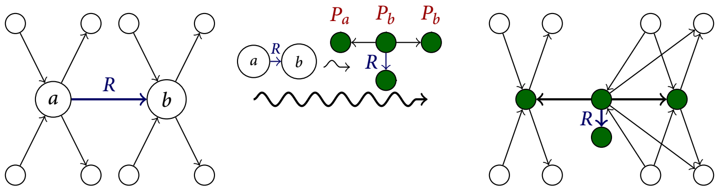
Relational Structures and Embeddings

$$\sigma \;:\quad \mathfrak{A} = (A, R_1^{\mathfrak{A}}, R_2^{\mathfrak{A}}, \ldots, R_k^{\mathfrak{A}}) \quad\rightarrow\quad (B, R_1^{\mathfrak{B}}, R_2^{\mathfrak{B}}, \ldots, R_k^{\mathfrak{B}}) = \mathfrak{B}$$

Embedding: σ is **injective** and $R_i^{\mathfrak{A}}(a_1, \ldots, a_{r_i}) \Leftrightarrow R_i^{\mathfrak{B}}(\sigma(a_1), \ldots, \sigma(a_{r_i}))$

Structure Rewriting Rules

Rewriting Example



Relational Structures and Embeddings

$$\sigma \;:\quad \mathfrak{A} = (A, R_1^{\mathfrak{A}}, R_2^{\mathfrak{A}}, \ldots, R_k^{\mathfrak{A}}) \quad\to\quad (B, R_1^{\mathfrak{B}}, R_2^{\mathfrak{B}}, \ldots, R_k^{\mathfrak{B}}) = \mathfrak{B}$$

Embedding: σ is **injective** and $R_i^{\mathfrak{A}}(a_1, \ldots, a_{r_i}) \Leftrightarrow R_i^{\mathfrak{B}}(\sigma(a_1), \ldots, \sigma(a_{r_i}))$

Rewriting Definition

$\mathfrak{B} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}/\sigma]$ iff $B = (A \setminus \sigma(L)) \dot\cup R$ and,
for $M = \{(r, a) \mid a = \sigma(l), r \in P_l^{\mathfrak{R}}$ for some $l \in L\} \cup \{(a, a) \mid a \in A\}$,

$$(b_1, \ldots, b_{r_i}) \in R_i^{\mathfrak{B}} \;\Leftrightarrow\; (b_1, \ldots, b_{r_i}) \in R_i^{\mathfrak{R}} \textbf{ or } (b_1 M \times \ldots \times b_{r_i} M) \cap R_i^{\mathfrak{A}} \neq \varnothing.$$

(in the second case at least one $b_j \notin \mathfrak{A}$)

Structure Rewriting Games

Game arena **(of a two-player zero-sum game)** is a **directed graph** with:

- vertices partitioned into positions of **Player 0** and **Player 1**
- edges **labelled by rewriting rules**

Structure Rewriting Games

Game arena **(of a two-player zero-sum game)** is a **directed graph** with:

- vertices partitioned into positions of **Player 0** and **Player 1**
- edges **labelled by rewriting rules**

Two interpretations of $\mathfrak{L} \to \mathfrak{R}$:

- **Existential**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}/\sigma]$, the player chooses the embedding σ
- **Universal**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}]$, **all** occurrences of $\mathfrak{L}$ are rewritten to $\mathfrak{R}$

Structure Rewriting Games

Game arena **(of a two-player zero-sum game)** is a **directed graph** with:

- vertices partitioned into positions of **Player 0** and **Player 1**
- edges **labelled by rewriting rules**

Two interpretations of $\mathfrak{L} \to \mathfrak{R}$:

- **Existential**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}/\sigma]$, the player chooses the embedding σ
- **Universal**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}]$, **all** occurrences of $\mathfrak{L}$ are rewritten to $\mathfrak{R}$

Winning conditions:

- L_μ (or temporal) formula ψ with **MSO sentences** for predicates, or
- MSO formula φ to be evaluated on the **limit** of the play
  Limit of $\mathfrak{A}_0\mathfrak{A}_1\mathfrak{A}_2 \ldots = \left(\bigcup_{n\in\mathbb{N}} \bigcap_{i\geq n} A_i,\ \bigcup_{n\in\mathbb{N}} \bigcap_{i\geq n} R^{\mathfrak{A}_i}\right)$
- **Reach** φ: Player 0 **wins** if the play reaches $\mathfrak{A}$ s.t. $\mathfrak{A} \models \varphi$

Structure Rewriting Games

Game arena (of a two-player zero-sum game) is a **directed graph** with:

- vertices partitioned into positions of **Player 0** and **Player 1**
- edges **labelled by rewriting rules**

Two interpretations of $\mathfrak{L} \to \mathfrak{R}$:

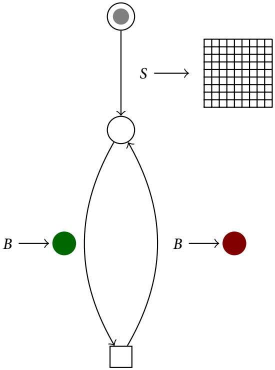
- **Existential**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}/\sigma]$, the player chooses the embedding σ
- **Universal**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}]$, **all** occurrences of $\mathfrak{L}$ are rewritten to $\mathfrak{R}$

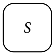
Winning conditions:

- L_μ (or temporal) formula ψ with **MSO sentences** for predicates, or
- MSO formula φ to be evaluated on the **limit** of the play
  Limit of $\mathfrak{A}_0\mathfrak{A}_1\mathfrak{A}_2\ldots = (\bigcup_{n\in\mathbb{N}}\bigcap_{i\geq n} A_i, \bigcup_{n\in\mathbb{N}}\bigcap_{i\geq n} R^{\mathfrak{A}_i})$
- **Reach φ**: Player 0 **wins** if the play reaches $\mathfrak{A}$ s.t. $\mathfrak{A} \models \varphi$

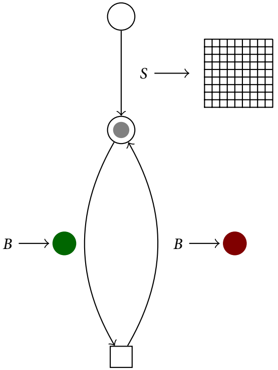
Motivation: many questions are **naturally defined as such games**:
constraint satisfaction, model checking, graph measures, games people play

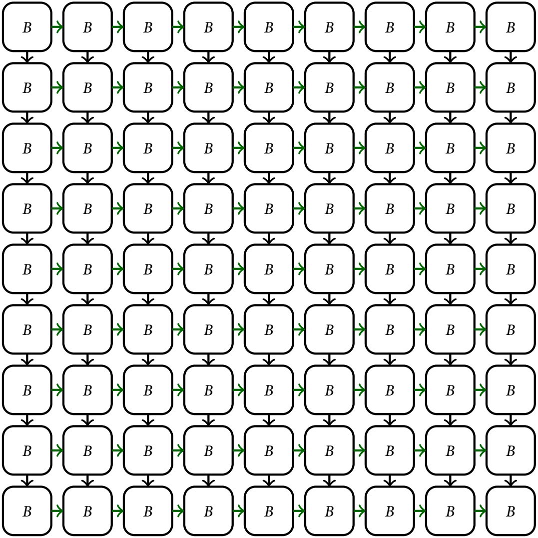
Example Game: Gomoku (Connect–5)



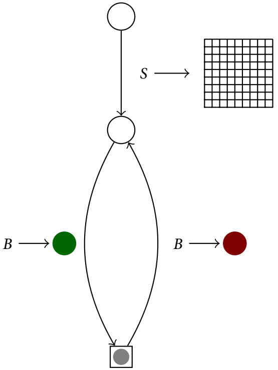


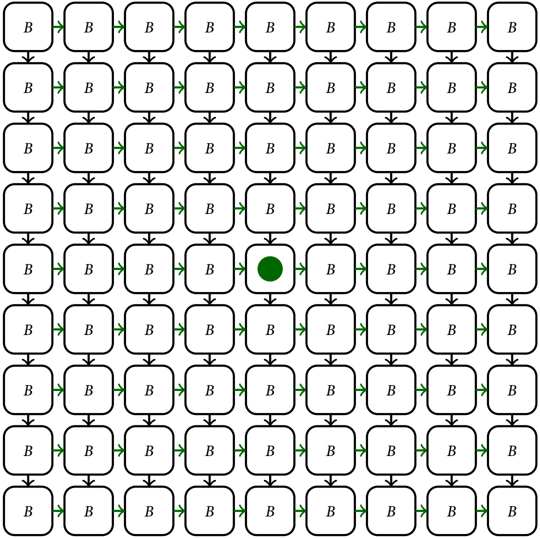
Example Game: Gomoku (Connect–5)



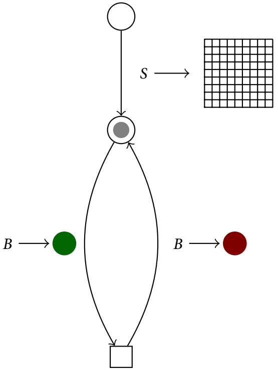


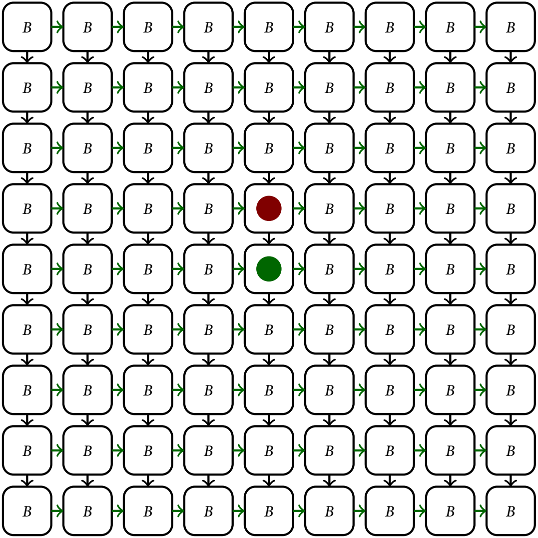
Example Game: Gomoku (Connect–5)



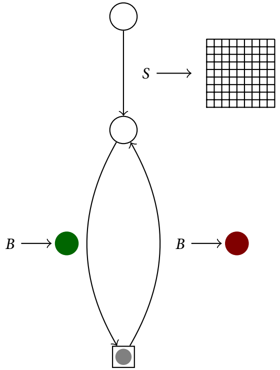


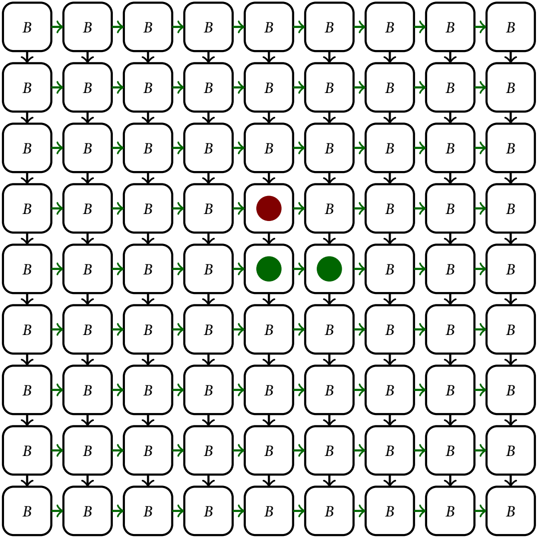
Example Game: Gomoku (Connect–5)



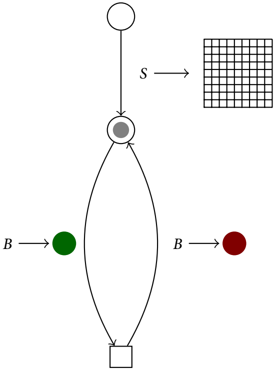


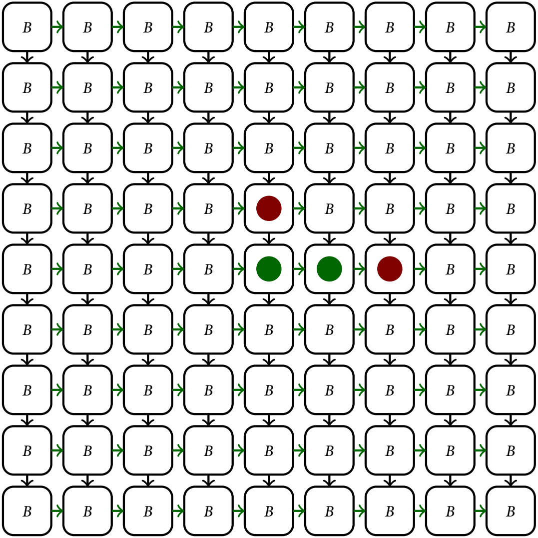
Example Game: Gomoku (Connect–5)



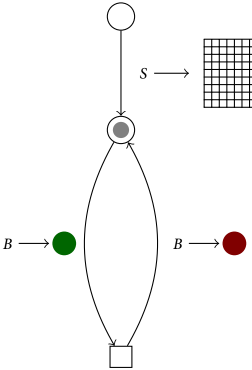


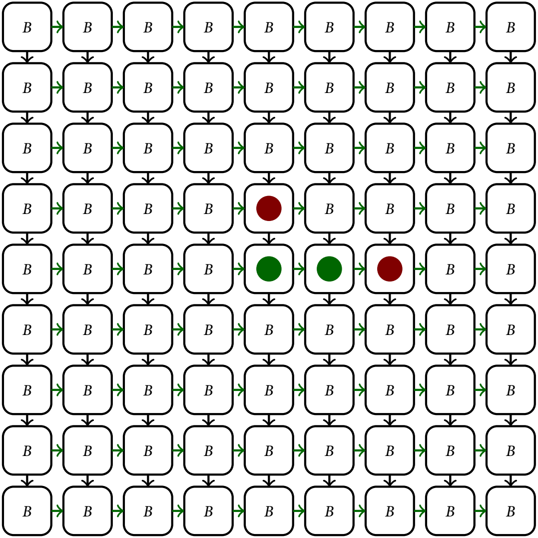
Example Game: Gomoku (Connect–5)





Example Game: Gomoku (Connect–5)





$$\exists x_1 \dots x_5 \Big(\bigwedge_{1\le i\le 5} G(x_i) \wedge \Big(\bigwedge_{1\le i\le 5} R(x_i, x_{i+1}) \vee \bigwedge_{1\le i\le 5} C(x_i, x_{i+1}) \vee \bigwedge_{1\le i\le 5} \exists y (R(x_i, y) \wedge C(y, x_{i+1})) \vee \bigwedge_{1\le i\le 5} \exists y (R(x_i, y) \wedge C(x_{i+1}, y)) \Big) \Big)$$

Overview

Structure Rewriting

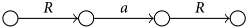
Separated Games

Simulation-Based Playing

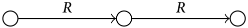
Simple Structure Rewriting

Separated Structures: no element is in two **non-terminal** relations
(Courcelle, Engelfriet, Rozenberg, 1991)

Separated:



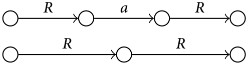
Not Separated:



Simple Structure Rewriting

Separated Structures: no element is in two **non-terminal** relations **(Courcelle, Engelfriet, Rozenberg, 1991)**

Separated:
Not Separated:

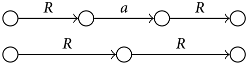


Simple Rule $\mathfrak{L} \to \mathfrak{R}$: $\mathfrak{R}$ is **separated** and $\mathfrak{L}$ is a **single tuple in relation**

Simple Structure Rewriting

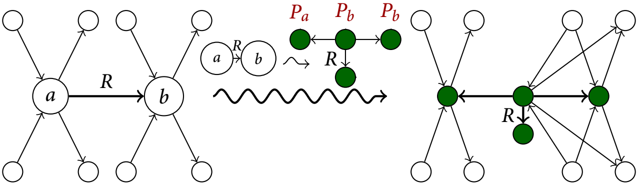
Separated Structures: no element is in two **non-terminal** relations
(Courcelle, Engelfriet, Rozenberg, 1991)

Separated:
Not Separated:



Simple Rule $\mathfrak{L} \to \mathfrak{R}$: $\mathfrak{R}$ is **separated** and $\mathfrak{L}$ is a **single tuple in relation**

Example



Decidability of Simple Rewriting Games

Logics

- L_μ[MSO]: Temporal properties expressed in L_μ (subsumes LTL) with properties of structures (states) expressed in MSO
- lim MSO: Property of the limit structure expressed in MSO

Theorem

- *Let R be a* ***finite*** *set of* ***(universal) simple structure rewriting rules,***
- *and φ be an L_μ[MSO] or lim MSO formula.*

Then the set $\{\pi \in R^\omega \ : \ (\lim)S(\pi) \vDash \varphi\}$ is ***ω-regular****.*

Corollary

Establishing the winner of (universal) finite simple rewriting games is decidable.
The winner has a winning strategy of a simple form.

Proof: Interpreting a Structure in a Tree

Description of how to build $\mathfrak{A}$ is a tree $\mathcal{T}(\mathfrak{A})$ with:

- **Leafs** of **different colours** $1 \dots k$
- $\oplus$ representing **disjoint sum**
- $i \leftarrow j$ to **change colour** of **all** nodes from i to j
- $e(i, j)$ to **add all pairs** of (i, j)-coloured nodes to e

Proof: Interpreting a Structure in a Tree

Description of how to build $\mathfrak{A}$ is a tree $\mathcal{T}(\mathfrak{A})$ with:

- **Leafs** of **different colours** $1 \dots k$
- $\oplus$ representing **disjoint sum**
- $i \leftarrow j$ to **change colour** of **all** nodes from i to j
- $e(i, j)$ to **add all pairs** of (i, j)-coloured nodes to e

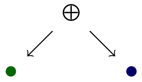
•

•

Proof: Interpreting a Structure in a Tree

Description of how to build $\mathfrak{A}$ is a tree $\mathcal{T}(\mathfrak{A})$ with:

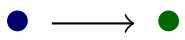
- **Leafs** of **different colours** $1 \ldots k$
- $\oplus$ representing **disjoint sum**
- $i \leftarrow j$ to **change colour** of **all** nodes from i to j
- $e(i, j)$ to **add all pairs** of (i, j)-coloured nodes to e

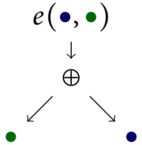


Proof: Interpreting a Structure in a Tree

Description of how to build $\mathfrak{A}$ is a tree $\mathcal{T}(\mathfrak{A})$ with:

- **Leafs** of **different colours** $1 \ldots k$
- $\oplus$ representing **disjoint sum**
- $i \leftarrow j$ to **change colour** of **all** nodes from i to j
- $e(i, j)$ to **add all pairs** of (i, j)-coloured nodes to e

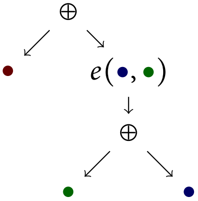




Proof: Interpreting a Structure in a Tree

Description of how to build $\mathfrak{A}$ is a tree $\mathcal{T}(\mathfrak{A})$ with:

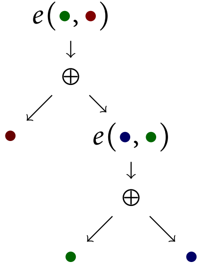
- **Leafs** of **different colours** $1 \ldots k$
- $\oplus$ representing **disjoint sum**
- $i \leftarrow j$ to **change colour** of **all** nodes from i to j
- $e(i, j)$ to **add all pairs** of (i, j)-coloured nodes to e



Proof: Interpreting a Structure in a Tree

Description of how to build $\mathfrak{A}$ is a tree $\mathcal{T}(\mathfrak{A})$ with:

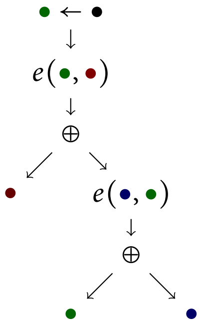
- **Leafs** of **different colours** $1 \ldots k$
- $\oplus$ representing **disjoint sum**
- $i \leftarrow j$ to **change colour** of **all** nodes from i to j
- $e(i, j)$ to **add all pairs** of (i, j)-coloured nodes to e



Proof: Interpreting a Structure in a Tree

Description of how to build $\mathfrak{A}$ is a tree $\mathcal{T}(\mathfrak{A})$ with:

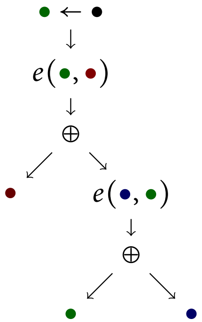
- **Leafs** of **different colours** $1 \dots k$
- $\oplus$ representing **disjoint sum**
- $i \leftarrow j$ to **change colour** of **all** nodes from i to j
- $e(i, j)$ to **add all pairs** of (i, j)-coloured nodes to e



Proof: Interpreting a Structure in a Tree

Description of how to build $\mathfrak{A}$ is a tree $\mathcal{T}(\mathfrak{A})$ with:

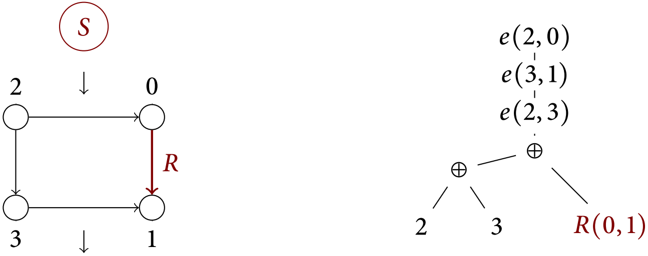
- **Leafs** of **different colours** $1 \dots k$
- $\oplus$ representing **disjoint sum**
- $i \leftarrow j$ to **change colour** of **all** nodes from i to j
- $e(i, j)$ to **add all pairs** of (i, j)-coloured nodes to e



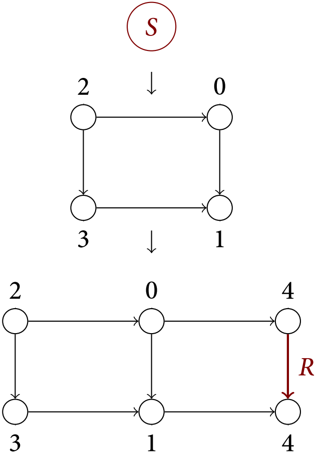
Theorem:
For every k there is an MSO-to-MSO **interpretation** $\mathcal{I}$ such that for all structures $\mathfrak{A}$ of **clique-width** $\leq k$ holds $\mathcal{I}(\mathcal{T}(\mathfrak{A})) \cong \mathfrak{A}$.

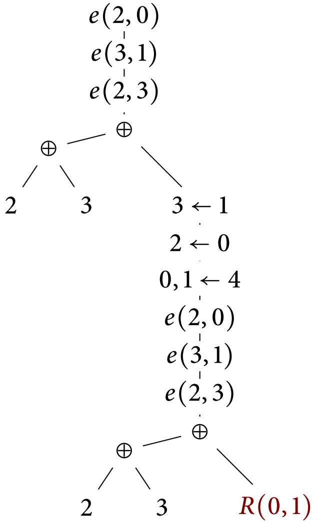
Proof: Simple Rewriting in the Tree

Proof: Simple Rewriting in the Tree

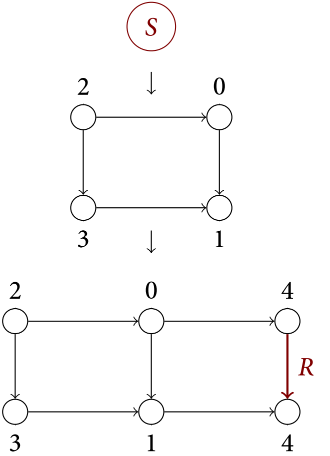


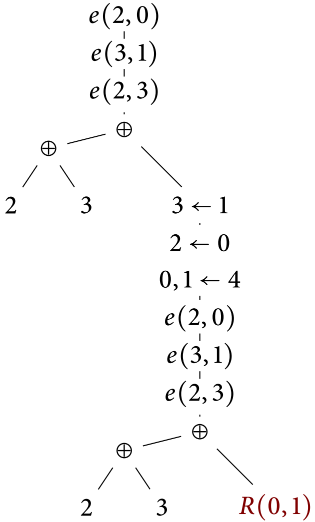
Proof: Simple Rewriting in the Tree





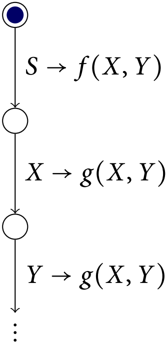
Proof: Simple Rewriting in the Tree



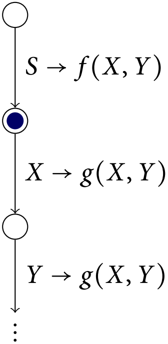


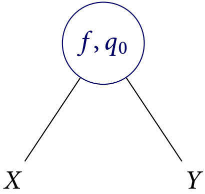
MSO-to-MSO interpretation: $\varphi \to \psi$

Proof: From Tree to Alternating Word Automata

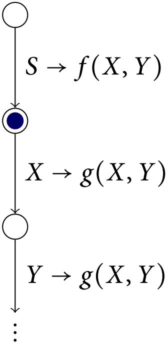


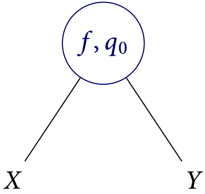
Proof: From Tree to Alternating Word Automata





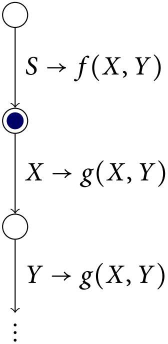
Proof: From Tree to Alternating Word Automata

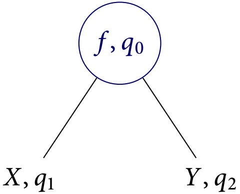




existential: pick transition

Proof: From Tree to Alternating Word Automata

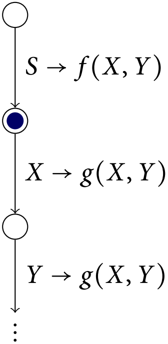


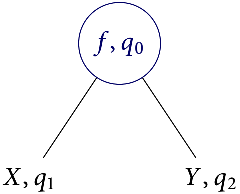


existential: pick transition

$f, q_0 \to (q_1, q_2)$

Proof: From Tree to Alternating Word Automata



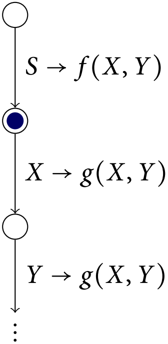


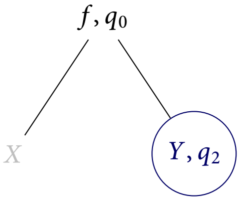
existential: pick transition

universal: left or right

$$f, q_0 \to (q_1, q_2)$$

Proof: From Tree to Alternating Word Automata



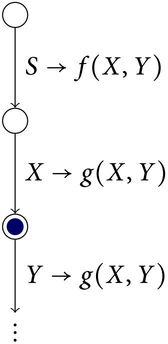


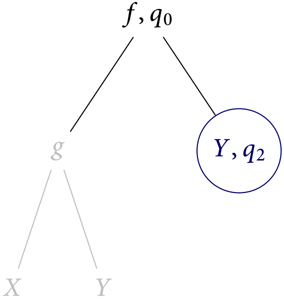
existential: pick transition

$f, q_0 \to (q_1, q_2)$

universal: left or right

Proof: From Tree to Alternating Word Automata





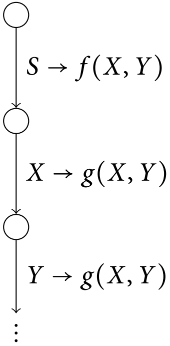
existential: pick transition

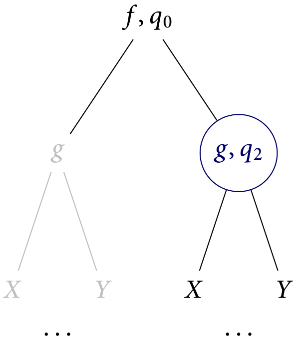
universal: left or right

$f, q_0 \to (q_1, q_2)$

ignore

Proof: From Tree to Alternating Word Automata





existential: pick transition

universal: left or right

$f, q_0 \to (q_1, q_2)$

ignore

Overview

Structure Rewriting

Separated Games

Simulation-Based Playing

Motivation

Implementing the Decidability Result

- **Tool**: **MONA**
  - Developed at **BRICS** since **1996** by Nils Klarlund and Anders Møller
  - Symbolic representation with **BDDs**
  - **Minimisation** at each step
- **Example:** a simple **tic-tac-toe** game $\rightsquigarrow$ **memory overflow**
- **Problems** due to **inefficient coding**
  - **Bounded clique-width graphs** not good for MONA
  - Only **universal interpretation** decidable, must encode games

Motivation

Implementing the Decidability Result

- **Tool**: **MONA**
  - Developed at **BRICS** since **1996** by Nils Klarlund and Anders Møller
  - Symbolic representation with **BDDs**
  - **Minimisation** at each step
- **Example:** a simple **tic-tac-toe** game $\rightsquigarrow$ **memory overflow**
- **Problems** due to **inefficient coding**
  - **Bounded clique-width graphs** not good for MONA
  - Only **universal interpretation** decidable, must encode games

Simulation-Based Game Playing (Joint work with Ł. Stafiniak)

Motivation

Implementing the Decidability Result

- **Tool**: **MONA**
  - Developed at **BRICS** since **1996** by Nils Klarlund and Anders Møller
  - Symbolic representation with **BDDs**
  - **Minimisation** at each step
- **Example:** a simple **tic-tac-toe** game $\rightsquigarrow$ **memory overflow**
- **Problems** due to **inefficient coding**
  - **Bounded clique-width graphs** not good for MONA
  - Only **universal interpretation** decidable, must encode games

Simulation-Based Game Playing (Joint work with Ł. Stafiniak)

How to determine the **value of a position** v in a **general** game?

- **Both players** play from v **randomly** a large number of times
- Calculate the **ratio of wins** of each player

Motivation

Implementing the Decidability Result

- **Tool**: **MONA**
  - Developed at **BRICS** since **1996** by Nils Klarlund and Anders Møller
  - Symbolic representation with **BDDs**
  - **Minimisation** at each step
- **Example:** a simple **tic-tac-toe** game $\rightsquigarrow$ **memory overflow**
- **Problems** due to **inefficient coding**
  - **Bounded clique-width graphs** not good for MONA
  - Only **universal interpretation** decidable, must encode games

Simulation-Based Game Playing (Joint work with Ł. Stafiniak)

How to determine the **value of a position** v in a **general** game?

- **Both players** play from v **randomly** a large number of times
- Calculate the **ratio of wins** of each player
- **Problem:** no way to **look forward** and **choose** actions

Upper Confidence Bounds for Trees

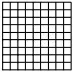
Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by the success of **MoGo**

Upper Confidence Bounds for Trees

Building a Tree during Random Plays

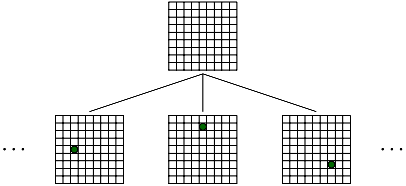
- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by the success of **MoGo**



Upper Confidence Bounds for Trees

Building a Tree during Random Plays

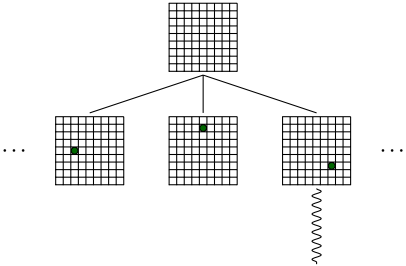
- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by the success of **MoGo**



Upper Confidence Bounds for Trees

Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by the success of **MoGo**



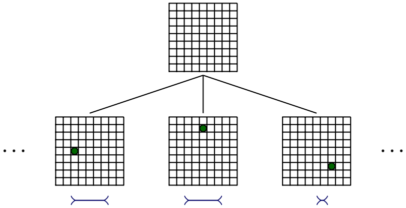
Upper Confidence Bounds for Trees

Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by the success of **MoGo**

Pick Max Upper Confidence

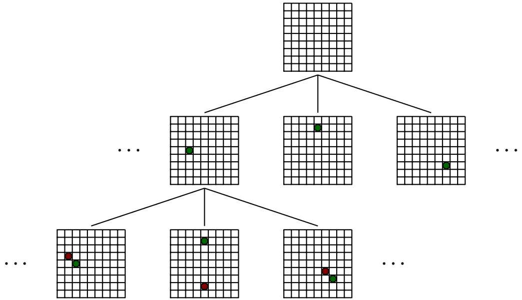
$$C \cdot \sqrt{\frac{\ln(n(v)+1)}{n(w)+1}}$$



Upper Confidence Bounds for Trees

Building a Tree during Random Plays

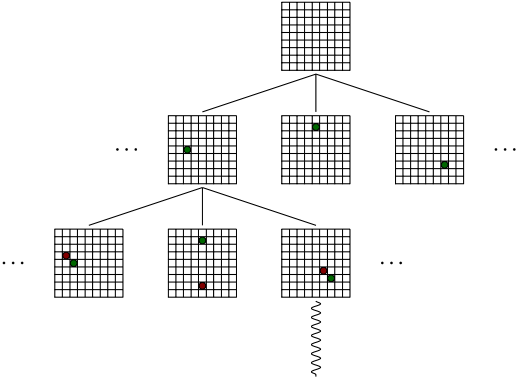
- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by the success of **MoGo**



Upper Confidence Bounds for Trees

Building a Tree during Random Plays

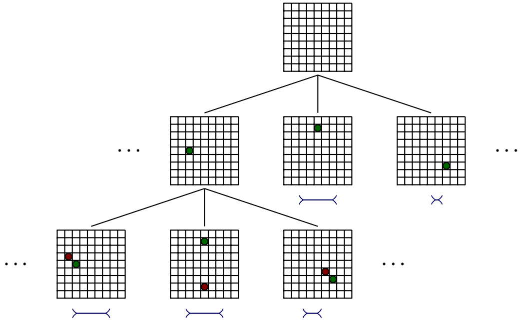
- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by the success of **MoGo**



Upper Confidence Bounds for Trees

Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by the success of **MoGo**



UCT for Structure Rewriting Games

Problems

- Fully random player is **too general**
- Large number of **formula evaluations** (slow)

UCT for Structure Rewriting Games

Problems

- Fully random player is **too general**
- Large number of **formula evaluations** (slow)

Improving Random Player

- **Hints: formulas** which separate **good and bad** states (**moves**)
- **Example (Gomoku):** connected group of stones is good
- **In MoGo: good and bad** patterns

UCT for Structure Rewriting Games

Problems

- Fully random player is **too general**
- Large number of **formula evaluations** (slow)

Improving Random Player

- **Hints: formulas** which separate **good and bad** states (**moves**)
- **Example (Gomoku):** connected group of stones is good
- **In MoGo: good and bad** patterns
- **Problem:** makes algorithm **even slower**

UCT for Structure Rewriting Games

Problems

- Fully random player is **too general**
- Large number of **formula evaluations** (slow)

Improving Random Player

- **Hints: formulas** which separate **good and bad** states (**moves**)
- **Example (Gomoku):** connected group of stones is good
- **In MoGo: good and bad** patterns
- **Problem:** makes algorithm **even slower**

Results of Hints

- **Breakthrough**: **beat if possible** ca. 70% improvement
- **Gomoku**: **play near your stone** ca. 80% improvement

UCT for Structure Rewriting Games

Problems

- Fully random player is **too general**
- Large number of **formula evaluations** (slow)

Improving Random Player

- **Hints: formulas** which separate **good and bad** states (**moves**)
- **Example (Gomoku):** connected group of stones is good
- **In MoGo: good and bad** patterns
- **Problem:** makes algorithm **even slower**

Results of Hints

- **Breakthrough**: **beat if possible** ca. 70% improvement
- **Gomoku**: **play near your stone** ca. 80% improvement

Perspective: once a good hint is found, prove that the strategy is **winning**.

How to Make the Solver Faster?

Solver Requirements

(1) **Obvious**: evaluate formulas fast

(2) **Repetition**: evaluate the same formula on **many structures**

(3) **Composition**: structures change **only slightly**

How to Make the Solver Faster?

Solver Requirements

(1) **Obvious**: evaluate formulas fast

(2) **Repetition**: evaluate the same formula on **many structures**

(3) **Composition**: structures change **only slightly**

MSO is compositional:

$$\mathrm{Th}^k(\mathfrak{A} \oplus^{\text{connect}} \mathfrak{B}) = \mathrm{Th}^k(\mathfrak{A}) \oplus^{\text{connect}} \mathrm{Th}^k(\mathfrak{B})$$

Using this requires **multiple CNF-DNF conversions**

How to Make the Solver Faster?

Solver Requirements

(1) **Obvious**: evaluate formulas fast

(2) **Repetition**: evaluate the same formula on **many structures**

(3) **Composition**: structures change **only slightly**

MSO is compositional:

$$\mathrm{Th}^k(\mathfrak{A} \oplus^{\text{connect}} \mathfrak{B}) = \mathrm{Th}^k(\mathfrak{A}) \oplus^{\text{connect}} \mathrm{Th}^k(\mathfrak{B})$$

Using this requires **multiple CNF-DNF conversions**

Current Solver Architecture (`toss.sourceforge.net`)

- **FO assignments**: represented directly
- **MSO assignments**: semi-symbolically

$$(1 \in X \wedge 2 \in X \wedge 3 \notin X) \vee (1 \notin X)$$

- Operations on MSO assignments: **use SAT solver**, **CNF-DNF again**
- Are **BDDs** better? **Unclear.**

Conclusions

Structure Rewriting Games

- **General** model of games with **structured states**
- Establishing the winner is **decidable** for **certain subclasses**
- **Simulation** can be used to **play** the games
- Possibly **learn formulas** from simulated plays

Conclusions

Structure Rewriting Games

- **General** model of games with **structured states**
- Establishing the winner is **decidable** for **certain subclasses**
- **Simulation** can be used to **play** the games
- Possibly **learn formulas** from simulated plays

Extensions

- **Preconditions and postconditions** in rewriting rules
- **Types of structures** (based on bounded clique-width)
- **Continuous dynamics** can be added
  - defined e.g. using **$\mathbb{R}$-structures and differential equations**
  - simple **quantitative logics** can be used

Conclusions

Structure Rewriting Games

- **General** model of games with **structured states**
- Establishing the winner is **decidable** for **certain subclasses**
- **Simulation** can be used to **play** the games
- Possibly **learn formulas** from simulated plays

Extensions

- **Preconditions and postconditions** in rewriting rules
- **Types of structures** (based on bounded clique-width)
- **Continuous dynamics** can be added
  - defined e.g. using **$\mathbb{R}$-structures and differential equations**
  - simple **quantitative logics** can be used

Thank You