Analyzing Structure Rewriting Systems

Łukasz Kaiser

Mathematische Grundlagen der Informatik
RWTH Aachen

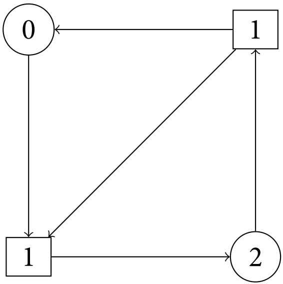
AlgoSyn

Aachen, 2009

Theory meets Practice

Theory meets Practice

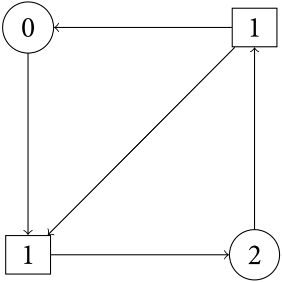
Look, I can do synthesis for that:



(Ł. Kaiser)

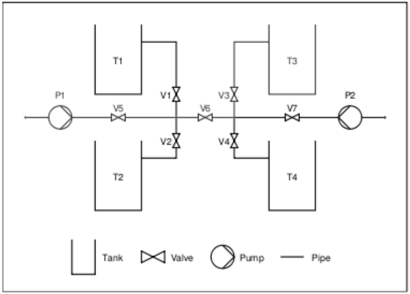
Theory meets Practice

Look, I can do synthesis for that:



(Ł. Kaiser)

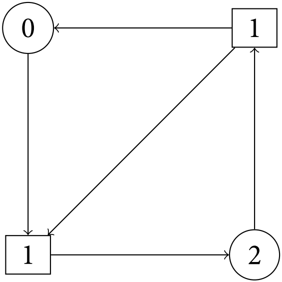
Great, I have something very similar:



(G. Quirós)

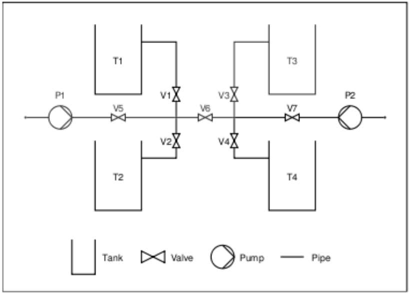
Theory meets Practice

Look, I can do synthesis for that:



(Ł. Kaiser)

Great, I have something very similar:

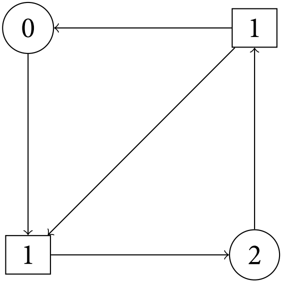


(G. Quirós)

But the whole right side is just **one state** on the left!

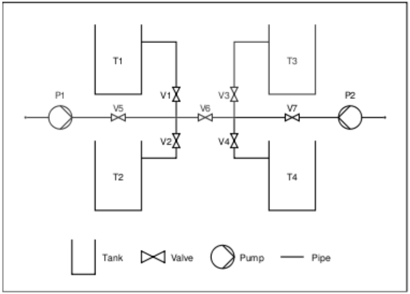
Theory meets Practice

Look, I can do synthesis for that:



(Ł. Kaiser)

Great, I have something very similar:



(G. Quirós)

But the whole right side is just one state on the left!

Can we model states by arbitrary relational structures?

Structure Rewriting Rules

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 $R_i^{\mathfrak{A}}(a_1, \ldots, a_{r_i}) \Leftrightarrow R_i^{\mathfrak{B}}(\sigma(a_1), \ldots, \sigma(a_{r_i}))$

Structure Rewriting Rules

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 $R_i^{\mathfrak{A}}(a_1, \ldots, a_{r_i}) \Leftrightarrow R_i^{\mathfrak{B}}(\sigma(a_1), \ldots, \sigma(a_{r_i}))$

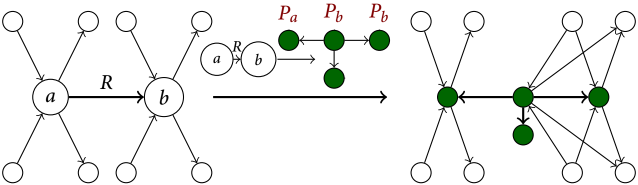
Rewriting Definition

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

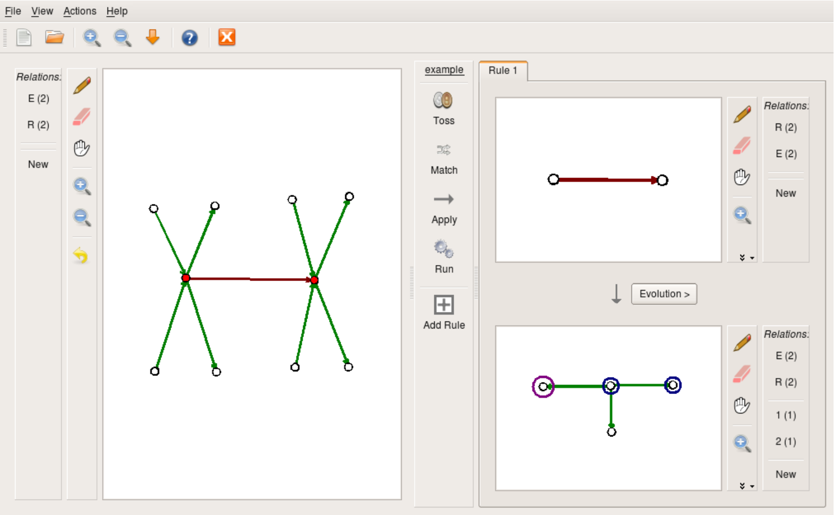
$$(b_1, \ldots, b_{r_i}) \in R_i^{\mathfrak{B}} \;\Leftrightarrow\; (b_1, \ldots, b_{r_i}) \in R_i^{\mathfrak{R}} \textbf{ or } (b_1 M \times \ldots \times b_{r_i} M) \cap R_i^{\mathfrak{A}} \neq \emptyset.$$

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

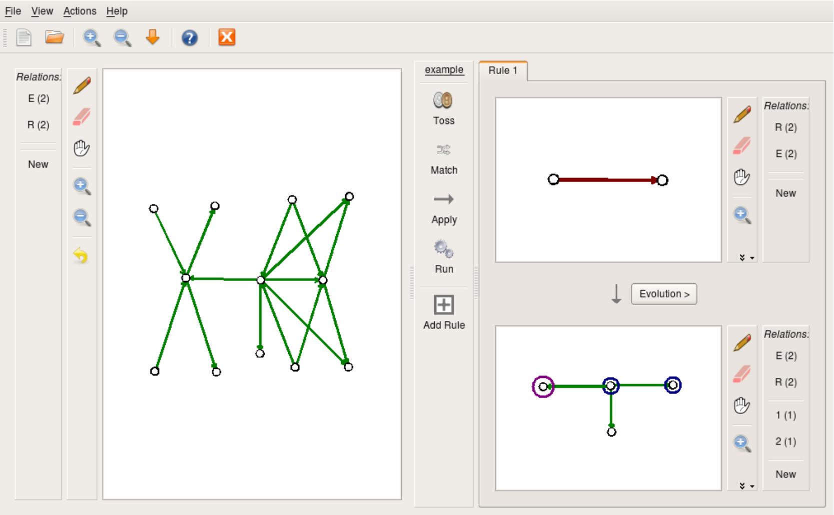
Rewriting Example



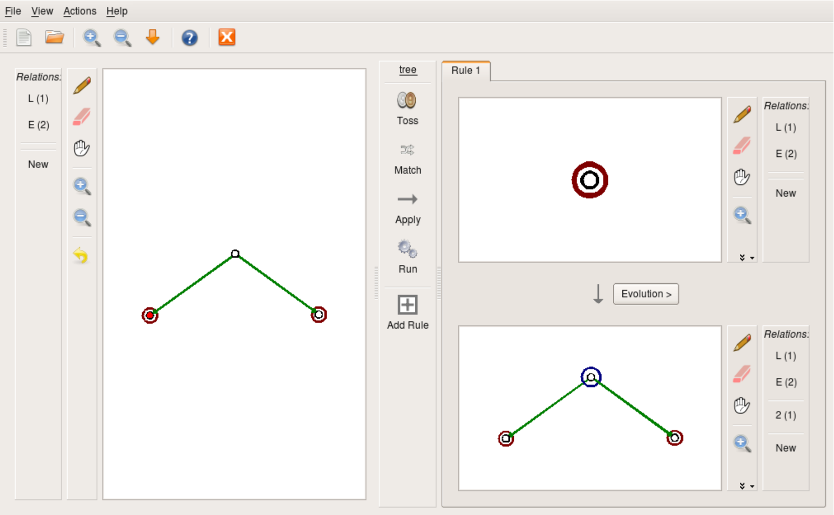
Structure Rewriting: Example



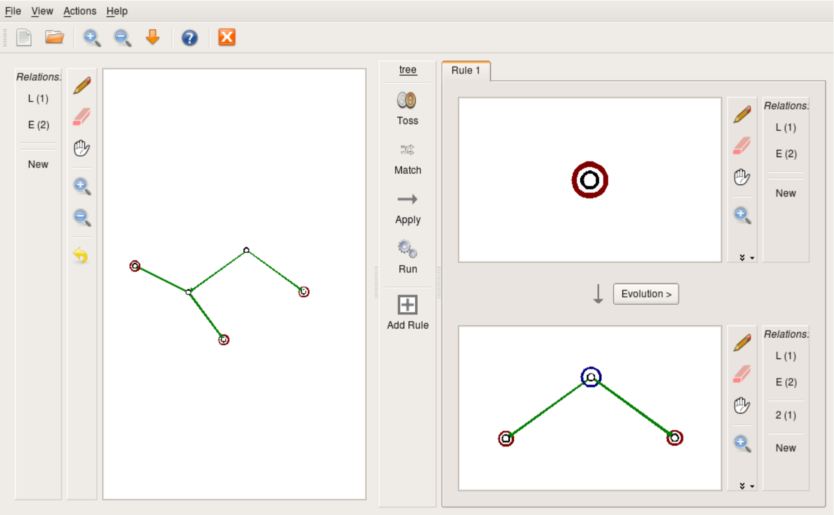
Structure Rewriting: Example



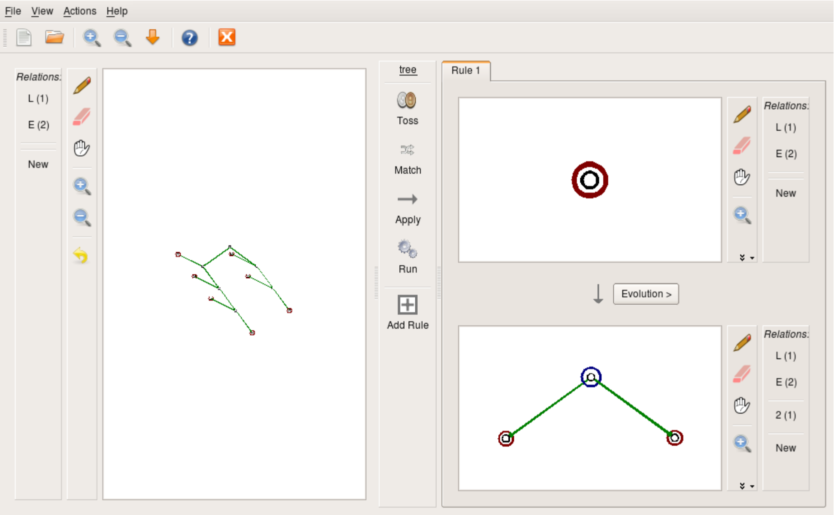
Structure Rewriting: Binary Tree



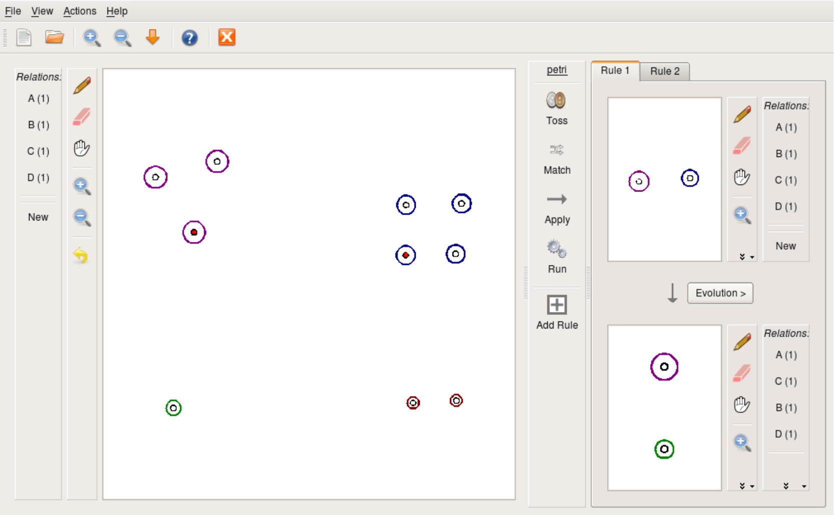
Structure Rewriting: Binary Tree



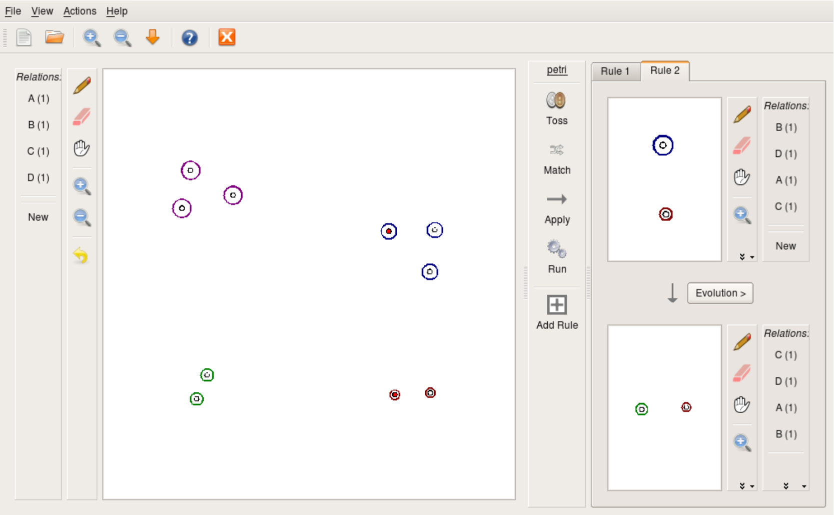
Structure Rewriting: Binary Tree



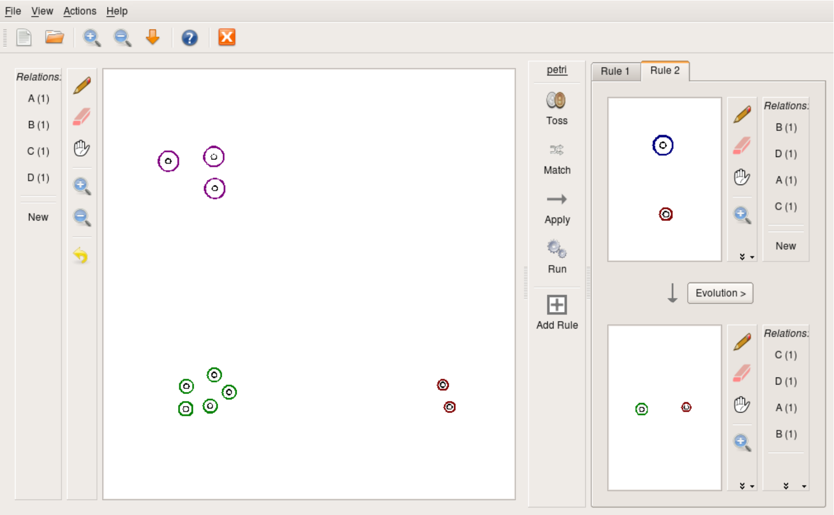
Structure Rewriting: Petri Net



Structure Rewriting: Petri Net

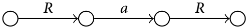


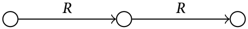
Structure Rewriting: Petri Net



Simple Structure Rewriting

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

Separated: 

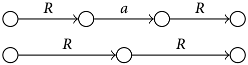
Not Separated: 

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**

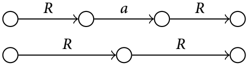
Universal Rewriting and Limit Structures

- $\mathfrak{A}[\mathfrak{L} \to \mathfrak{R}]$ is $\mathfrak{A}$ with **all** occurrences of $\mathfrak{L}$ rewritten to $\mathfrak{R}$
- **Limit** of $\mathfrak{A}_0 \to \mathfrak{A}_1 \to \mathfrak{A}_2 \to \dots$: $(\bigcup_{n\in\mathbb{N}} \bigcap_{i\geq n} A_i, \bigcup_{n\in\mathbb{N}} \bigcap_{i\geq n} R_i)$

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**

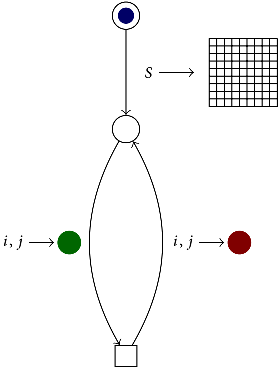
Universal Rewriting and Limit Structures

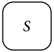
- $\mathfrak{A}[\mathfrak{L} \to \mathfrak{R}]$ is $\mathfrak{A}$ with **all** occurrences of $\mathfrak{L}$ rewritten to $\mathfrak{R}$
- **Limit** of $\mathfrak{A}_0 \to \mathfrak{A}_1 \to \mathfrak{A}_2 \to \ldots$: $\left(\bigcup_{n\in\mathbb{N}} \bigcap_{i\geq n} A_i,\ \bigcup_{n\in\mathbb{N}} \bigcap_{i\geq n} R_i\right)$

Why Universal Rewriting for Games?

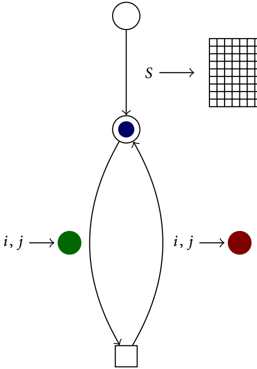
- **In contrast** to **graph grammars** (single player)
- Establishing the winner if players **pick** embeddings is **undecidable:**
  - simulate **active context-free games** (thanks to **Anca Muscholl**)
- Choosing embedding can be allowed for **bounded number of non-terminals**

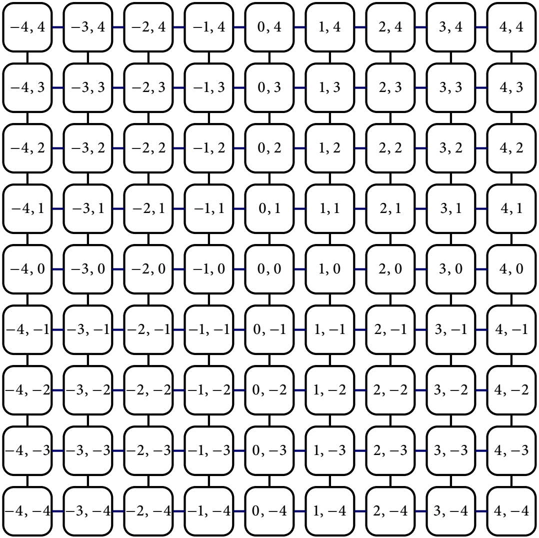
Example Game Played with Structures (Gomoku)



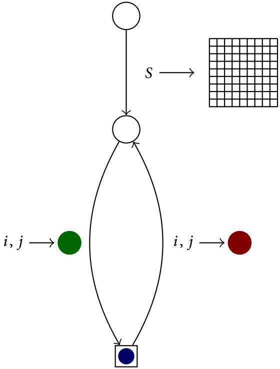


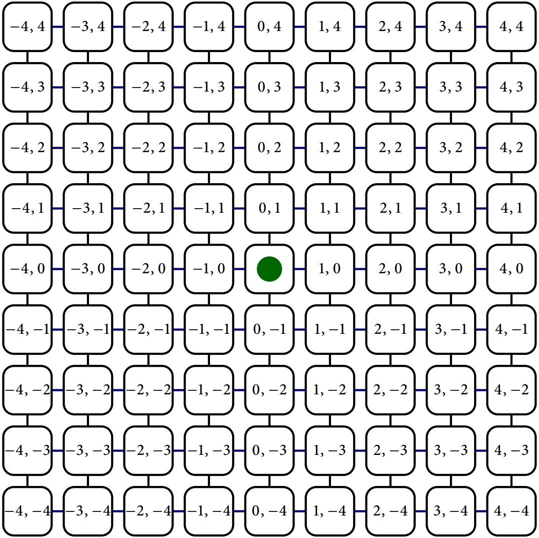
Example Game Played with Structures (Gomoku)



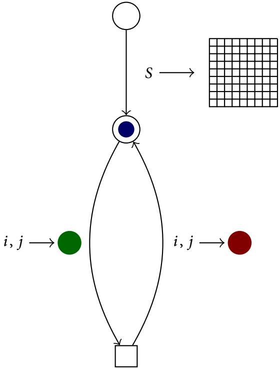


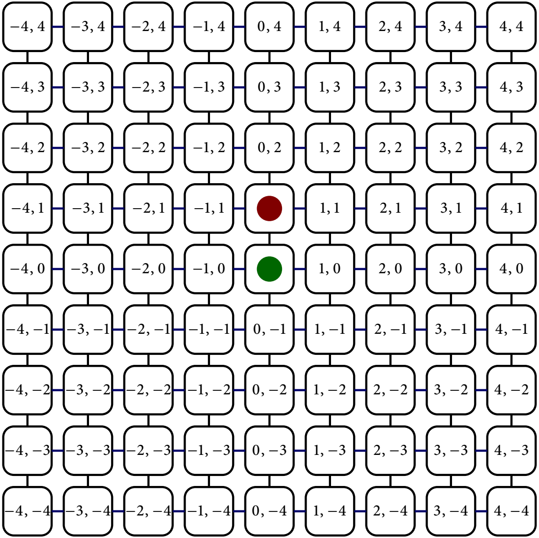
Example Game Played with Structures (Gomoku)



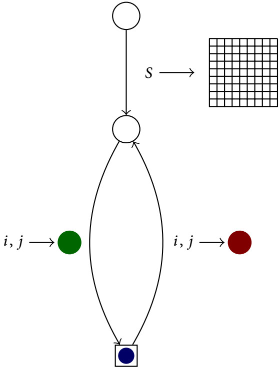


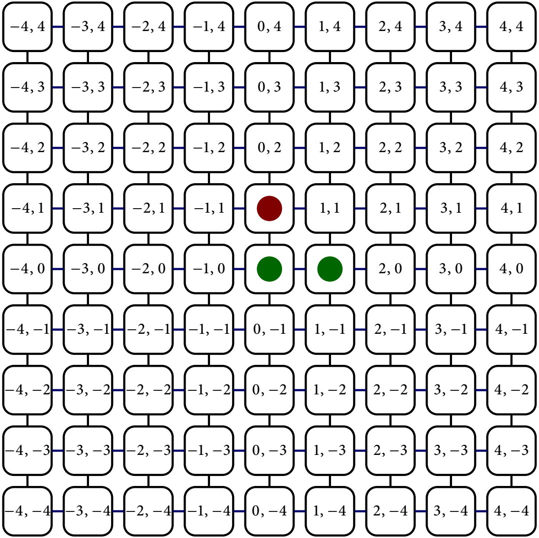
Example Game Played with Structures (Gomoku)



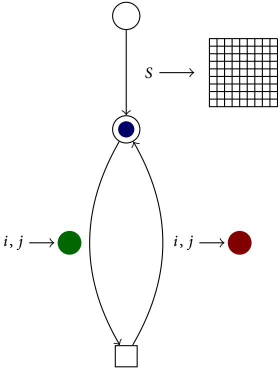


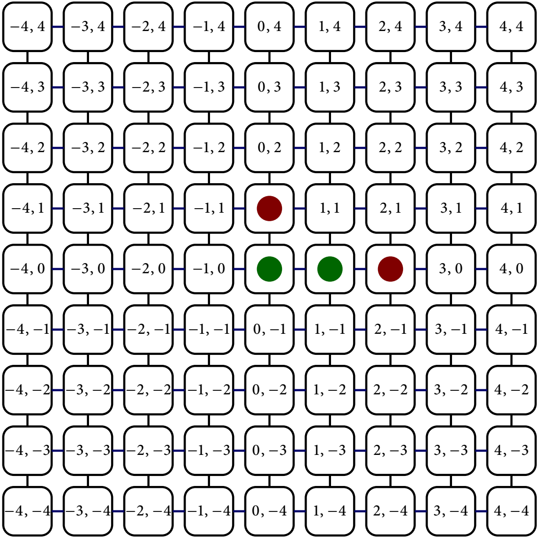
Example Game Played with Structures (Gomoku)





Example Game Played with Structures (Gomoku)





Main Result

Logics

- Properties of structures (states) expressed in MSO
- Temporal properties expressed in **the modal μ-calculus, L_μ**
- **Alternatively:** property of the limit structure expressed in MSO

Theorem

- *Let R be a **finite** set of **simple separated structure rewriting rules***
- *and φ be an $\mathsf{L}_\mu[\mathsf{MSO}]$ (or MSO) formula giving the **winning condition***

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

Corollary

Establishing the winner of finite separated rewriting games is decidable.

Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \dots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect** **all** nodes of colour i to **all** of colour j

Example:

Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \dots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect** **all** nodes of colour i to **all** of colour j

Example:

Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \dots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect** **all** nodes of colour i to **all** of colour j

Example:

Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \ldots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect** **all** nodes of colour i to **all** of colour j

Example:

Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \ldots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect** **all** nodes of colour i to **all** of colour j

Example:

Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \ldots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect** **all** nodes of colour i to **all** of colour j

Example:

Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \dots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect** **all** nodes of colour i to **all** of colour j

Example:

Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \dots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect all** nodes of colour i to **all** of colour j

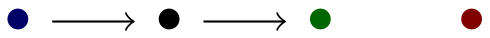
Example:

Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \ldots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect** **all** nodes of colour i to **all** of colour j

Example:

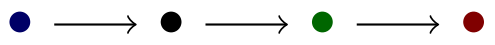


Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \ldots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect all** nodes of colour i to **all** of colour j

Example:



Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \ldots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect all** nodes of colour i to **all** of colour j

Example:

$$\bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \cdots \longrightarrow \bullet$$

Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \dots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect** **all** nodes of colour i to **all** of colour j

Example:

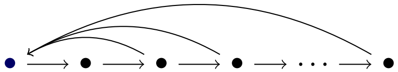
$$\bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \cdots \longrightarrow \bullet$$

Proof: How to Build a Graph

Pieces to build a graph:

- Bags of **single nodes** with **different colours** $1 \ldots K$
- Paint to **change colour** of **all** nodes from i to j
- Edges to **connect** **all** nodes of colour i to **all** of colour j

Example:



Proof: Interpreting a Graph in a Tree

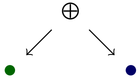
Description of how to build $\mathcal{G}$ is a tree $\mathcal{T}(\mathcal{G})$:

Proof: Interpreting a Graph in a Tree

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

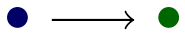
Proof: Interpreting a Graph in a Tree

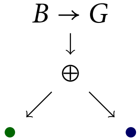
Description of how to build $\mathcal{G}$ is a tree $\mathcal{T}(\mathcal{G})$:



Proof: Interpreting a Graph in a Tree

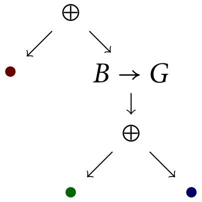
Description of how to build $\mathcal{G}$ is a tree $\mathcal{T}(\mathcal{G})$:





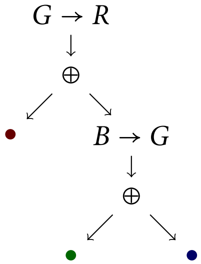
Proof: Interpreting a Graph in a Tree

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



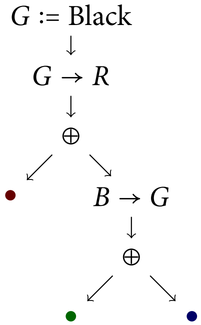
Proof: Interpreting a Graph in a Tree

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



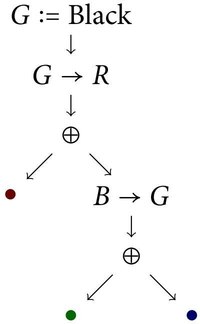
Proof: Interpreting a Graph in a Tree

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



Proof: Interpreting a Graph in a Tree

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



Theorem:
For every K there is an MSO-to-MSO **interpretation** $\mathcal{I}$ such that for all graphs $\mathcal{G}$ of **clique-width** $\leq K$ holds

$$\mathcal{I}(\mathcal{T}(\mathcal{G})) \cong \mathcal{G}$$

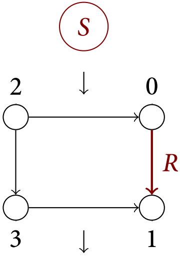
Proof: Separated Rewriting as a Tree

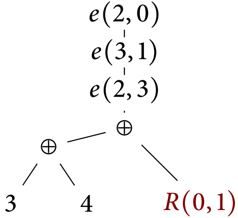
S

$\downarrow$

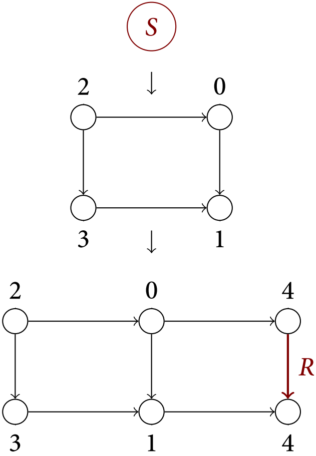
S

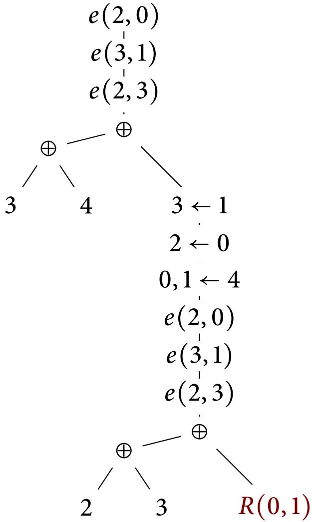
Proof: Separated Rewriting as a Tree



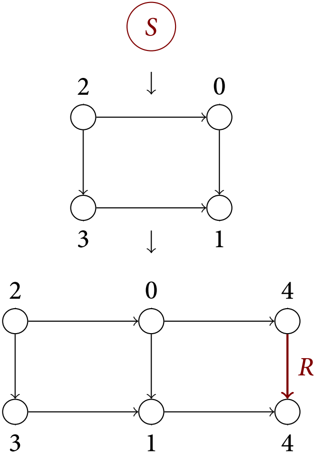


Proof: Separated Rewriting as a Tree

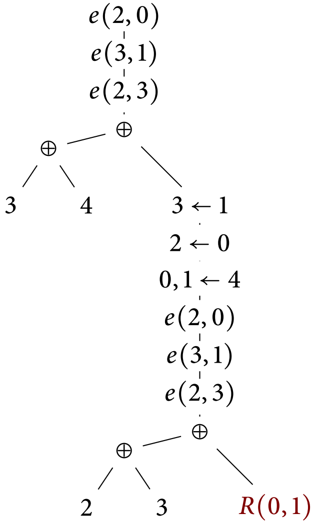




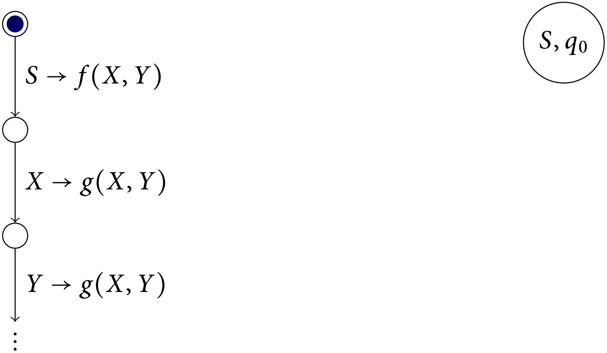
Proof: Separated Rewriting as a Tree



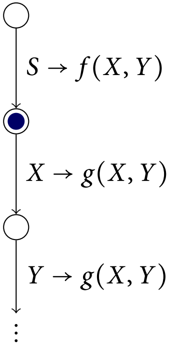
MSO-to-MSO **interpretation:** $\varphi \to \psi$

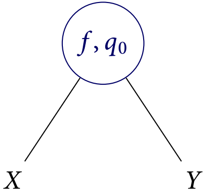


Proof: From Tree to Alternating Word Automata

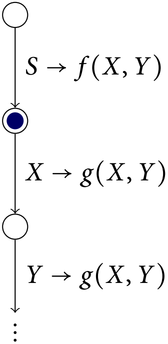


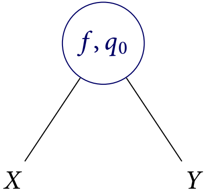
Proof: From Tree to Alternating Word Automata





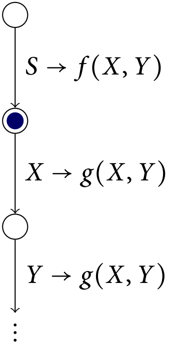
Proof: From Tree to Alternating Word Automata

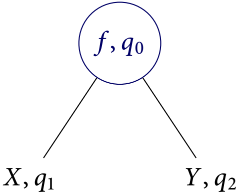




existential: pick transition

Proof: From Tree to Alternating Word Automata

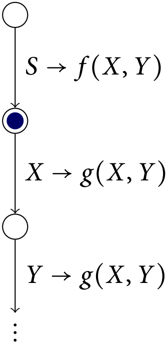


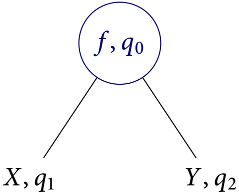


existential: pick transition

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

Proof: From Tree to Alternating Word Automata



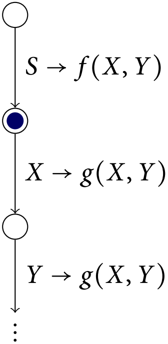


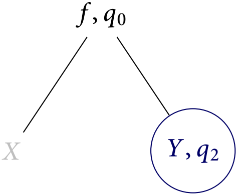
existential: pick transition

universal: left or right

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

Proof: From Tree to Alternating Word Automata



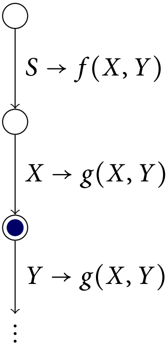


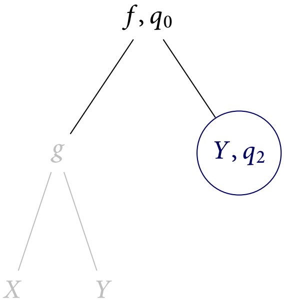
existential: pick transition

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

universal: left or right

Proof: From Tree to Alternating Word Automata





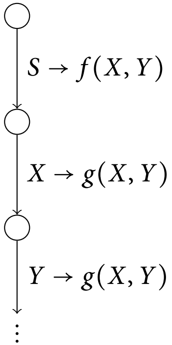
existential: pick transition

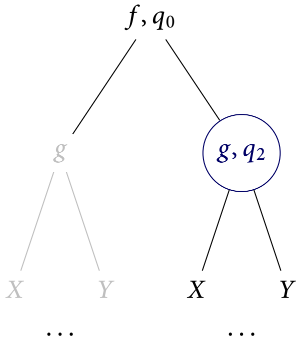
universal: left or right

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

ignore

Proof: From Tree to Alternating Word Automata





existential: pick transition

universal: left or right

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

ignore

Outlook

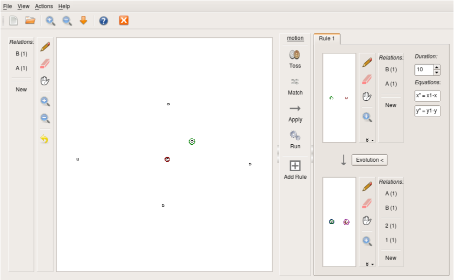
Structure Rewriting Extensions

- The way of **combining sides of a rule** can be generalised

Outlook

Structure Rewriting Extensions

- The way of **combining sides of a rule** can be generalised
- **Continuous dynamics** can be added
  - defined e.g. using **$\mathbb{R}$-structures and differential equations**
  - simple **quantitative logics** can be used



Outlook

Structure Rewriting Extensions

- The way of **combining sides of a rule** can be generalised
- **Continuous dynamics** can be added
  - defined e.g. using **$\mathbb{R}$-structures and differential equations**
  - simple **quantitative logics** can be used

Analyzing Structure Rewriting Systems

- **Result: synthesis** possible for **separated rewriting games**
- The theorem on **separated games can be generalised:**
  - to anything known about ***ω*-regular games**
  - to **some infinite arenas** e.g. pushdown graphs
- **Unary predicates left and right: Petri Nets**, generalisations?
- Other **logics and corresponding graph measures**, e.g. FO, FO[Reach]**?**
- **General method:** reduction to **trees**. Do **term rewriting methods** apply?
- Can we use **abstraction** for more complex rewriting systems?

Outlook

Structure Rewriting Extensions

- The way of **combining sides of a rule** can be generalised
- **Continuous dynamics** can be added
  - defined e.g. using **$\mathbb{R}$-structures and differential equations**
  - simple **quantitative logics** can be used

Analyzing Structure Rewriting Systems

- **Result: synthesis** possible for **separated rewriting games**
- The theorem on **separated games can be generalised:**
  - to anything known about **ω-regular games**
  - to **some infinite arenas** e.g. pushdown graphs
- **Unary predicates left and right: Petri Nets**, generalisations?
- Other **logics and corresponding graph measures**, e.g. FO, FO[Reach]**?**
- **General method:** reduction to **trees**. Do **term rewriting methods** apply?
- Can we use **abstraction** for more complex rewriting systems?

Thank You