

# ANALYZING STRUCTURE REWRITING SYSTEMS

Łukasz Kaiser

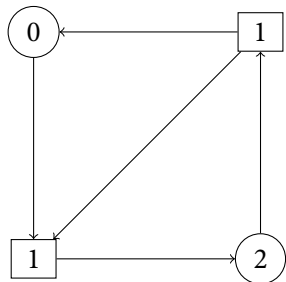
Mathematische Grundlagen der Informatik  
RWTH Aachen

**ALGO**SYN  
Aachen, 2009

# THEORY MEETS PRACTICE

# THEORY MEETS PRACTICE

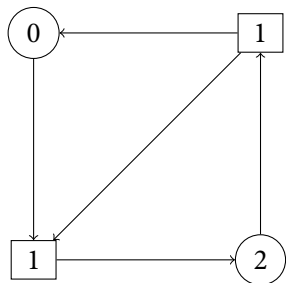
Look, I can do synthesis for that:



(Ł. Kaiser)

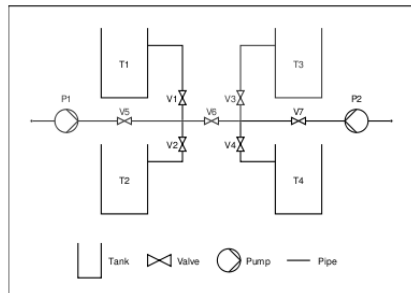
# THEORY MEETS PRACTICE

Look, I can do synthesis for that:



(Ł. Kaiser)

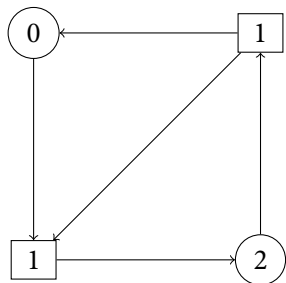
Great, I have something very **similar**:



(G. Quirós)

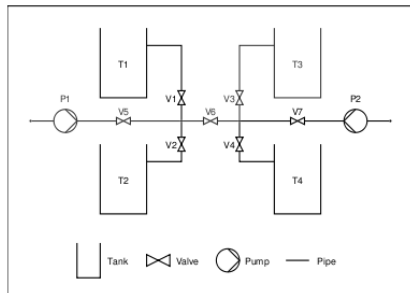
# THEORY MEETS PRACTICE

Look, I can do synthesis for that:



(Ł. Kaiser)

Great, I have something very **similar**:

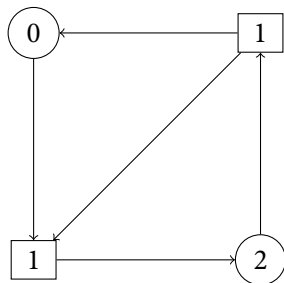


(G. Quirós)

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

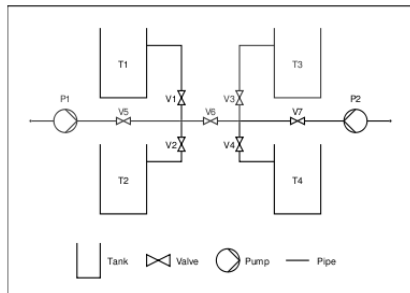
# THEORY MEETS PRACTICE

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

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

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

# STRUCTURE REWRITING RULES

## Relational Structures and Embeddings

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

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

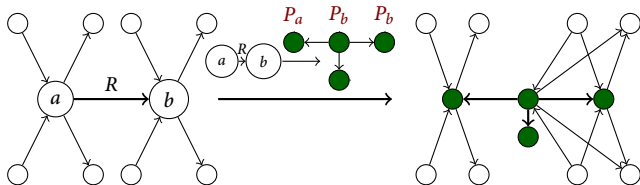
## Rewriting Definition

$\mathfrak{B} = \mathfrak{A}[\mathcal{L} \rightarrow \mathfrak{R}/\sigma]$  iff  $B = (A \setminus \sigma(L)) \dot{\cup} R$  and,

for  $M = \{(r, a) \mid a = \sigma(l), r \in P_l^{\mathfrak{R}} \text{ for some } l \in L\} \cup \{(a, a) \mid a \in A\}$ ,

$(b_1, \dots, b_{r_i}) \in R_i^{\mathfrak{B}} \Leftrightarrow (b_1, \dots, b_{r_i}) \in R_i^{\mathfrak{A}} \text{ or } (b_1 M \times \dots \times b_{r_i} M) \cap R_i^{\mathfrak{A}} \neq \emptyset$   
(in the second case at least one  $b_j \notin \mathfrak{A}$ )

## Rewriting Example





# STRUCTURE REWRITING: EXAMPLE

The screenshot displays a software interface for structure rewriting, divided into several panels:

- Top Panel:** Contains a menu bar with "File", "View", "Actions", and "Help", and a toolbar with icons for file operations, search, and help.
- Left Panel:** Labeled "Relations:", it contains a list of relations: "E (2)", "R (2)", and "New". Below the list are icons for editing, deleting, and creating relations.
- Main Canvas (Left):** Shows a graph with two red nodes connected by a horizontal red edge. Each red node is also connected to two white nodes by green edges, forming two 'X' shapes.
- Right Panel (example):** Labeled "example", it contains a list of relations: "Toss", "Match", "Apply", and "Run". Below the list is an "Add Rule" button.
- Right Panel (Rule 1):** Labeled "Rule 1", it shows the transformation process. The top part shows the initial graph (two red nodes connected by a red edge). Below it is an "Evolution >" button. The bottom part shows the resulting graph after the rule is applied: the two red nodes are now blue, and the horizontal edge is green. The left blue node is also connected to a white node by a green edge.
- Right Panel (Relations):** Below the "Rule 1" panel, there are two "Relations:" panels. The top one lists "R (2)" and "E (2)". The bottom one lists "1 (1)" and "2 (1)".

# STRUCTURE REWRITING: EXAMPLE

The screenshot displays a software interface for structure rewriting, organized into several panels:

- Main Canvas:** A large central area showing a graph with 8 nodes and 12 green edges. The graph consists of a central horizontal edge connecting two nodes, with various other edges branching out from these nodes.
- Left Panel (Relations):** A sidebar with a list of relations: "E (2)", "R (2)", and "New". Below the list are icons for editing (pencil), deleting (eraser), moving (hand), zooming (magnifying glass), and undo (curved arrow).
- Right Panel (Rule 1):** A panel titled "Rule 1" showing the transformation process. It contains two sub-canvas areas:
  - Top Sub-canvas:** Shows the initial state of the rule, a single horizontal red edge connecting two white nodes.
  - Bottom Sub-canvas:** Shows the result of the rule application, a graph with 5 nodes and 5 green edges. The nodes are colored: the leftmost node is purple, the middle node is blue, and the rightmost node is blue. The edges are green.
- Central Control Panel:** A vertical strip of controls between the main canvas and the rule panel, including:
  - example:** A tab label.
  - Toss:** An icon of a coin.
  - Match:** An icon of two overlapping shapes.
  - Apply:** An arrow icon.
  - Run:** A gear icon.
  - Add Rule:** A plus sign icon.
- Evolution & Lists:** A downward arrow labeled "Evolution >" is positioned between the two sub-canvas areas. To the right of each sub-canvas is a "Relations:" list with icons for editing, deleting, moving, and zooming. The top list contains "R (2)" and "E (2)", while the bottom list contains "E (2)", "R (2)", "1 (1)", and "2 (1)".

# STRUCTURE REWRITING: BINARY TREE

The screenshot shows a software interface for structure rewriting on a binary tree. The interface is divided into several panels:

- Top Menu:** File, View, Actions, Help.
- Toolbar:** Contains icons for file operations (document, folder), search (magnifying glass), zoom (arrow), help (question mark), and close (X).
- Left Panel (Relations):** Contains a list of relations: L (1), E (2), and New. Below the list are icons for editing (pencil), deleting (eraser), moving (hand), and zooming (magnifying glass).
- Main Canvas (Left):** Displays a binary tree structure with a root node (white circle) and two leaf nodes (red circles with white centers), connected by green lines.
- Right Panel (Rule 1):** Contains a list of relations: L (1), E (2), and New. Below the list are icons for editing, deleting, moving, and zooming. A central canvas shows a target symbol (a red circle with a white center and a black ring). Below the canvas is a button labeled "Evolution >".
- Bottom Panel (Add Rule):** Contains a plus sign icon and the text "Add Rule".
- Bottom Panel (Relations):** Contains a list of relations: L (1), E (2), 2 (1), and New. Below the list are icons for editing, deleting, moving, and zooming.

The interface is designed for defining and applying rules to rewrite the structure of a binary tree. The "Evolution >" button suggests a sequence of transformations.

# STRUCTURE REWRITING: BINARY TREE

The screenshot shows a software interface for structure rewriting on a binary tree. The interface is divided into several panels:

- Top Menu:** File, View, Actions, Help.
- Toolbar:** Contains icons for file operations (document, folder), zooming (magnifying glass), and other actions (down arrow, question mark, close).
- Left Panel (Relations):** Contains a list of relations: L (1), E (2), and New. Below the list are icons for editing (pencil), deleting (eraser), moving (hand), and zooming (magnifying glass).
- Main Canvas (Left):** Displays a binary tree structure with green edges and red circular nodes. The root node is white, and the leaf nodes are red with a white center.
- Right Panel (Rule 1):** Contains a list of relations: L (1), E (2), and New. Below the list are icons for editing, deleting, moving, and zooming. A central canvas shows a red circular node with a white center, representing the target of the rule. Below this canvas is a button labeled "Evolution >".
- Bottom Panel (Add Rule):** Contains a plus sign icon and the text "Add Rule".
- Bottom Panel (Main Canvas):** Displays the same binary tree structure as the left panel, but with a blue circular node at the root position, indicating the result of the rule application.

# STRUCTURE REWRITING: BINARY TREE

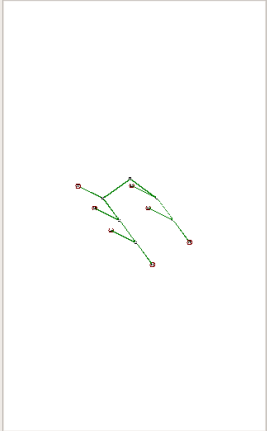
File View Actions Help

📄 📁 🔍 🔍 ⬇️ ? ✖️

**Relations:**

- L (1)
- E (2)
- New

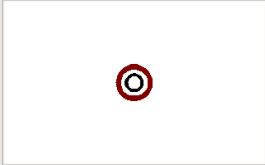
📏 ✍️ 🖱️ 🔍 🔍 ↶



**tree**

- Toss
- Match
- Apply
- Run
- Add Rule

**Rule 1**

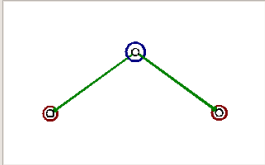


**Relations:**

- L (1)
- E (2)
- New

📏 ✍️ 🖱️ 🔍

↓ Evolution >



**Relations:**

- L (1)
- E (2)
- 2 (1)
- New

📏 ✍️ 🖱️ 🔍

# STRUCTURE REWRITING: PETRI NET

The screenshot displays a software interface for Petri net structure rewriting. The main workspace contains a Petri net with four places (circles) and one transition (circle with a dot). The places are labeled A, B, C, and D, each containing one token. The transition is labeled 'New' and contains one token. The interface includes a menu bar (File, View, Actions, Help) and a toolbar with icons for file operations, zooming, and a close button.

On the left, a 'Relations:' panel lists the places: A (1), B (1), C (1), and D (1), along with a 'New' button. A vertical toolbar on the left contains icons for drawing, erasing, moving, zooming, and undo.

On the right, a 'petri' panel contains buttons for 'Toss', 'Match', 'Apply', and 'Run', along with an 'Add Rule' button. Below this is a 'Rule 1' / 'Rule 2' panel. 'Rule 1' shows a Petri net with places A and B, each containing one token. 'Rule 2' shows a Petri net with places A and C, each containing one token. An 'Evolution >' button is located between the two rule panels. To the right of the rule panels, another 'Relations:' panel lists the places: A (1), B (1), C (1), and D (1), along with a 'New' button.

# STRUCTURE REWRITING: PETRI NET

The screenshot displays a software interface for Petri net manipulation. The main workspace contains a Petri net with several places and transitions. Places are represented by circles with a central dot, and transitions are represented by circles with a central cross. The places contain tokens, shown as small circles with a central dot.

**Main Petri Net:**

- Place A (top left): 1 token (purple)
- Place B (middle left): 1 token (purple)
- Place C (middle right): 1 token (purple)
- Place D (bottom left): 2 tokens (green)
- Transition 1 (top right): 1 token (blue)
- Transition 2 (middle right): 1 token (blue)
- Transition 3 (bottom right): 2 tokens (red)

**Left Panel (Relations):**

- A (1)
- B (1)
- C (1)
- D (1)
- New

**Right Panel (Rule 2):**

**Rule 2 Petri Net:**

- Place A (top): 1 token (blue)
- Place B (bottom): 1 token (red)

**Rule 2 Relations:**

- B (1)
- D (1)
- A (1)
- C (1)
- New

**Bottom Panel (Rule 1):**

**Rule 1 Petri Net:**

- Place C (top): 1 token (green)
- Place D (bottom): 1 token (red)

**Rule 1 Relations:**

- C (1)
- D (1)
- A (1)
- B (1)

**Central Panel (Actions):**

- Toss
- Match
- Apply
- Run
- Add Rule

**Evolution:** Evolution >

# STRUCTURE REWRITING: PETRI NET

The screenshot displays a software interface for Petri net simulation. The main window shows a Petri net with purple tokens (A, B, C, D) and green tokens (A, B, C, D). The interface includes a menu bar (File, View, Actions, Help), a toolbar with icons for file operations and navigation, and a left sidebar with a 'Relations' list and a 'New' button. The right sidebar contains a 'petri' section with icons for 'Toss', 'Match', 'Apply', 'Run', and 'Add Rule'. A rule application window is open, showing 'Rule 1' and 'Rule 2'. 'Rule 1' shows a blue token (A) and a red token (B). 'Rule 2' shows a green token (C) and a red token (B). The 'Evolution >' button is visible between the two rule windows. The 'Relations' list on the right of the rule application window shows the current state of the Petri net.

File View Actions Help

Relations:

- A (1)
- B (1)
- C (1)
- D (1)
- New

petri

- Toss
- Match
- Apply
- Run
- Add Rule

Rule 1 Rule 2

Relations:

- B (1)
- D (1)
- A (1)
- C (1)
- New

Evolution >

Relations:

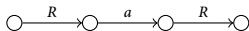
- C (1)
- D (1)
- A (1)
- B (1)



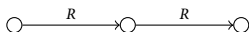
# SIMPLE STRUCTURE REWRITING

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

**Separated:**



**Not Separated:**

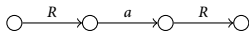


**Simple Rule  $\mathcal{L} \rightarrow \mathfrak{R}$ :**  $\mathfrak{R}$  is **separated** and  $\mathcal{L}$  is a **single tuple in relation**

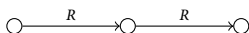
# SIMPLE STRUCTURE REWRITING

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

**Separated:**



**Not Separated:**



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

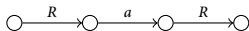
**Universal Rewriting and Limit Structures**

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

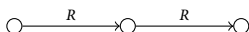
# SIMPLE STRUCTURE REWRITING

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

**Separated:**



**Not Separated:**



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

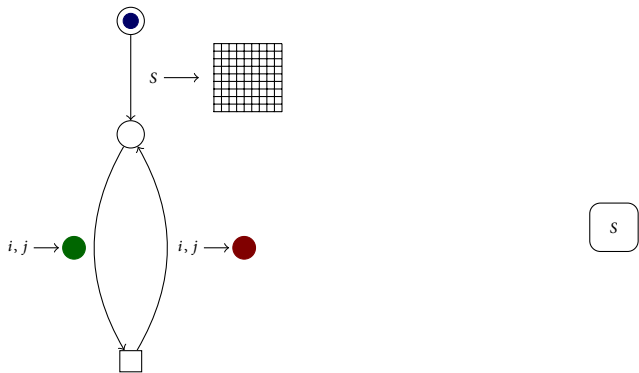
Universal Rewriting and Limit Structures

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

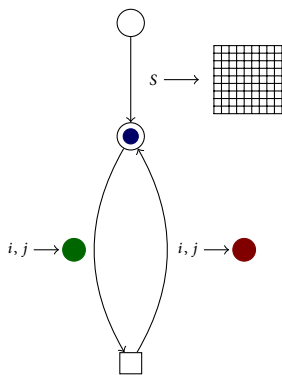
Why Universal Rewriting for Games?

- **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)

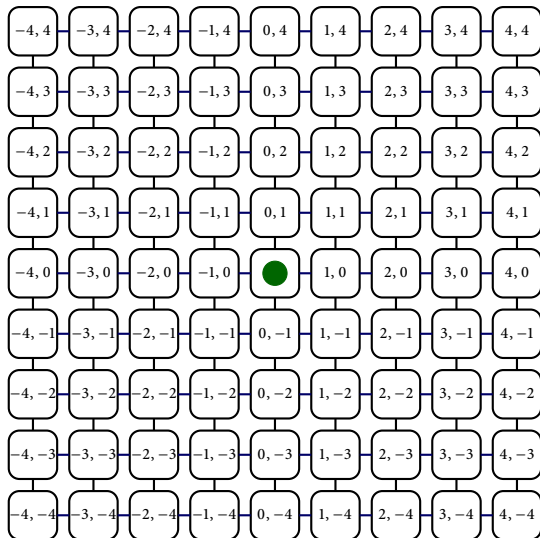
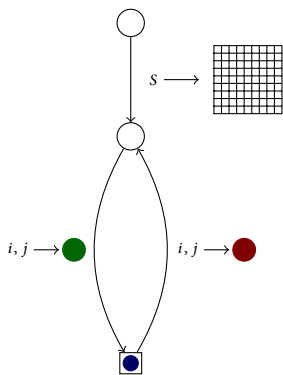


# EXAMPLE GAME PLAYED WITH STRUCTURES (GOMOKU)

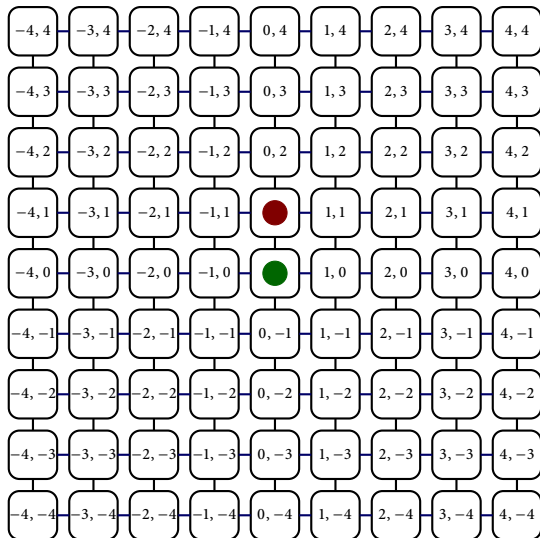
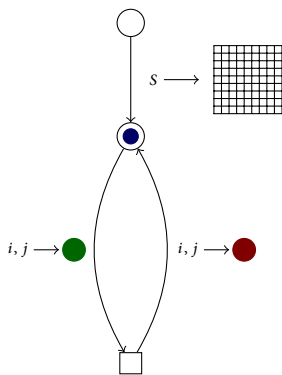


-4, 4	-3, 4	-2, 4	-1, 4	0, 4	1, 4	2, 4	3, 4	4, 4
-4, 3	-3, 3	-2, 3	-1, 3	0, 3	1, 3	2, 3	3, 3	4, 3
-4, 2	-3, 2	-2, 2	-1, 2	0, 2	1, 2	2, 2	3, 2	4, 2
-4, 1	-3, 1	-2, 1	-1, 1	0, 1	1, 1	2, 1	3, 1	4, 1
-4, 0	-3, 0	-2, 0	-1, 0	0, 0	1, 0	2, 0	3, 0	4, 0
-4, -1	-3, -1	-2, -1	-1, -1	0, -1	1, -1	2, -1	3, -1	4, -1
-4, -2	-3, -2	-2, -2	-1, -2	0, -2	1, -2	2, -2	3, -2	4, -2
-4, -3	-3, -3	-2, -3	-1, -3	0, -3	1, -3	2, -3	3, -3	4, -3
-4, -4	-3, -4	-2, -4	-1, -4	0, -4	1, -4	2, -4	3, -4	4, -4

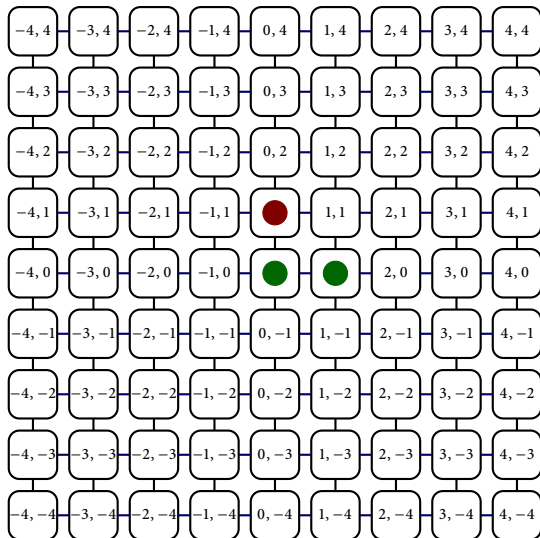
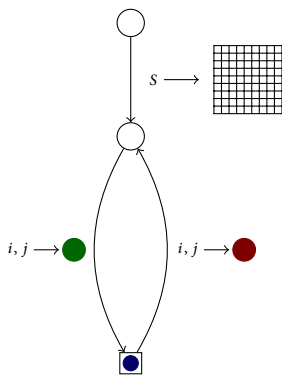
# EXAMPLE GAME PLAYED WITH STRUCTURES (GOMOKU)



# EXAMPLE GAME PLAYED WITH STRUCTURES (GOMOKU)

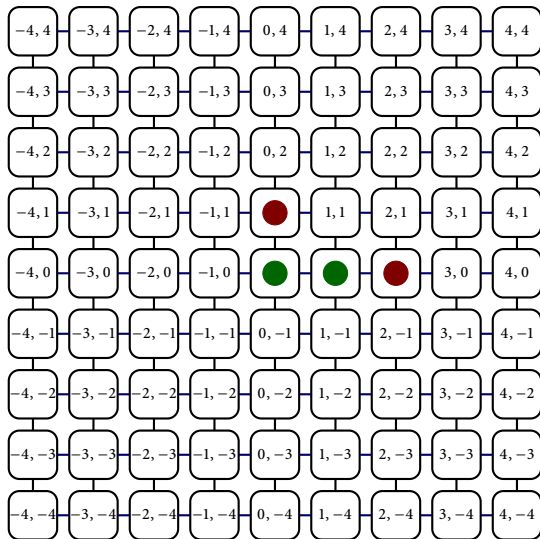
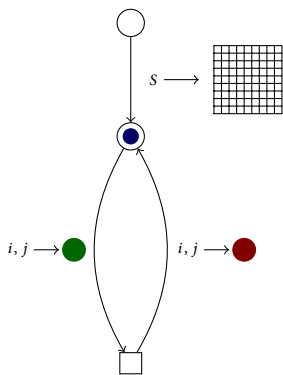


# EXAMPLE GAME PLAYED WITH STRUCTURES (GOMOKU)





# EXAMPLE GAME PLAYED WITH STRUCTURES (GOMOKU)



# MAIN RESULT

## Logics

- Properties of structures (states) expressed in **MSO**
- Temporal properties expressed in **the modal  $\mu$ -calculus,  $L_\mu$**
- **Alternatively:** property of the limit structure expressed in **MSO**

## Theorem

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

Then the set  $\{\pi \in R^\omega : (\text{lim})S(\pi) \models \varphi\}$  is  **$\omega$ -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**  $1 \dots K$
- 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**  $1 \dots K$
- 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**  $1 \dots K$
- 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**  $1 \dots K$
- 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: INTERPRETING A GRAPH IN A TREE

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



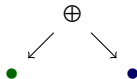
## PROOF: INTERPRETING A GRAPH IN A TREE

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



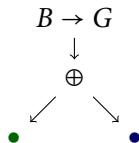
# PROOF: INTERPRETING A GRAPH IN A TREE

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



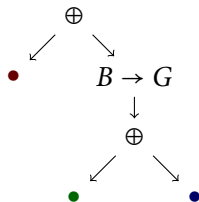
# PROOF: INTERPRETING A GRAPH IN A TREE

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



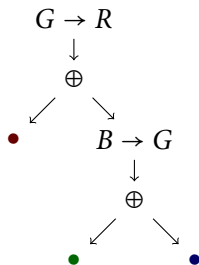
# PROOF: INTERPRETING A GRAPH IN A TREE

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



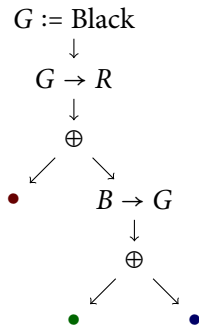
# PROOF: INTERPRETING A GRAPH IN A TREE

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



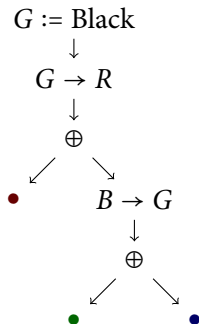
# PROOF: INTERPRETING A GRAPH IN A TREE

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

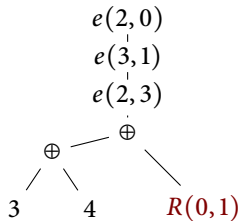
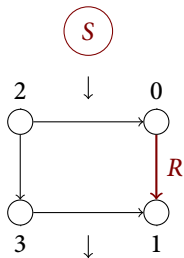
S



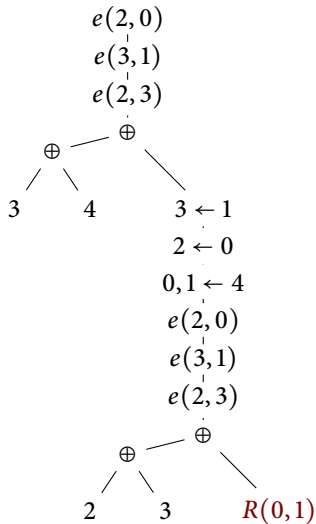
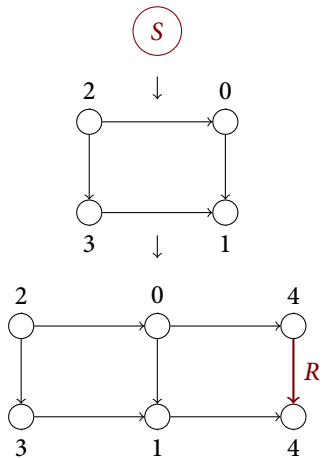
S



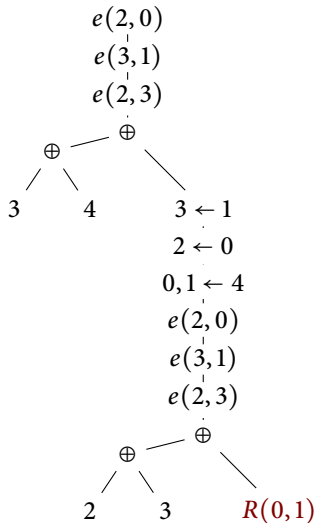
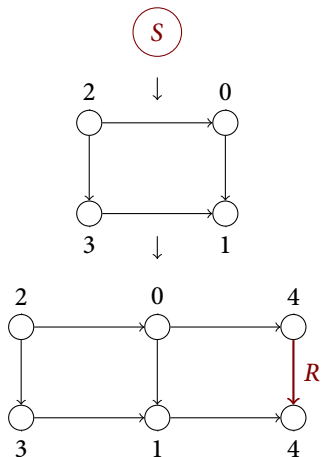
## PROOF: SEPARATED REWRITING AS A TREE



# PROOF: SEPARATED REWRITING AS A TREE

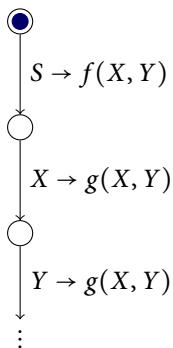


# PROOF: SEPARATED REWRITING AS A TREE

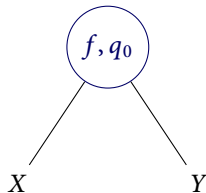
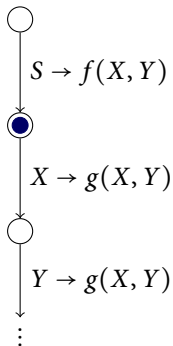


MSO-to-MSO interpretation:  $\varphi \rightarrow \psi$

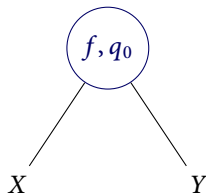
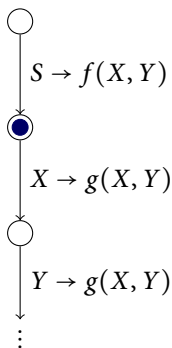
# PROOF: FROM TREE TO ALTERNATING WORD AUTOMATA



# PROOF: FROM TREE TO ALTERNATING WORD AUTOMATA

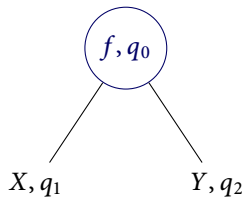
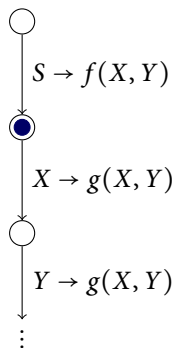


# PROOF: FROM TREE TO ALTERNATING WORD AUTOMATA



**existential:** pick transition

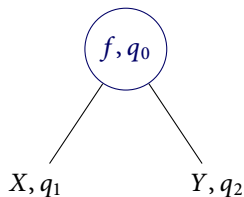
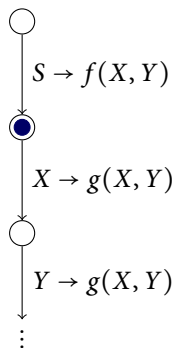
# PROOF: FROM TREE TO ALTERNATING WORD AUTOMATA



**existential:** pick transition

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

# PROOF: FROM TREE TO ALTERNATING WORD AUTOMATA



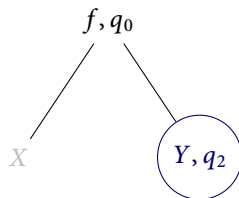
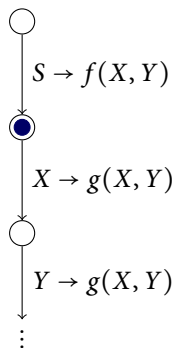
**existential:** pick transition

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

**universal:** left or right



# PROOF: FROM TREE TO ALTERNATING WORD AUTOMATA

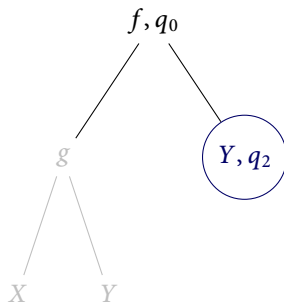
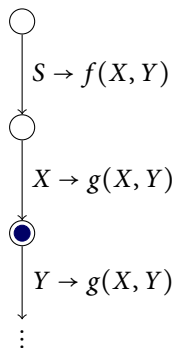


**existential:** pick transition

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

**universal:** left or right

# PROOF: FROM TREE TO ALTERNATING WORD AUTOMATA



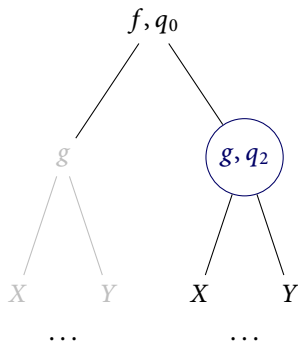
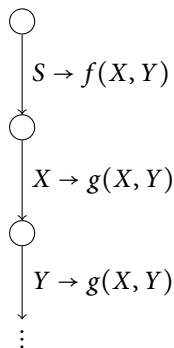
**existential:** pick transition

**universal:** left or right

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

**ignore**

# PROOF: FROM TREE TO ALTERNATING WORD AUTOMATA



**existential:** pick transition

**universal:** left or right

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

ignore

# OUTLOOK

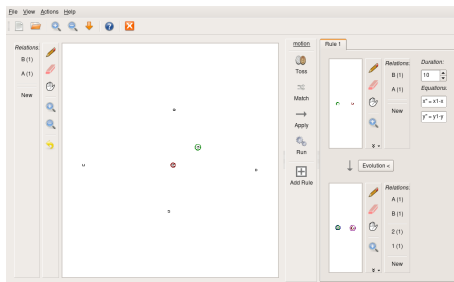
## Structure Rewriting Extensions

- 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  **$\omega$ -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  **$\omega$ -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