Structure Rewriting Games

Łukasz Kaiser

Mathematische Grundlagen der Informatik
RWTH Aachen

Automata and Algorithmic Logic
Stuttgart, 2009

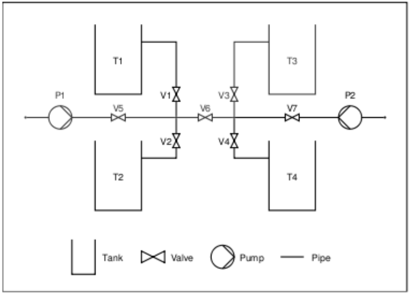
Motivation

AlgoSyn: Algorithmic Synthesis of Reactive and Discrete-Continuous Systems

Motivation

AlgoSyn: Algorithmic Synthesis of Reactive and Discrete-Continuous Systems

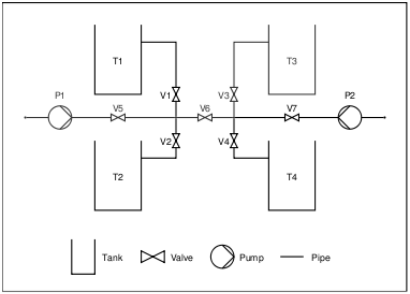
Part of a pumping station



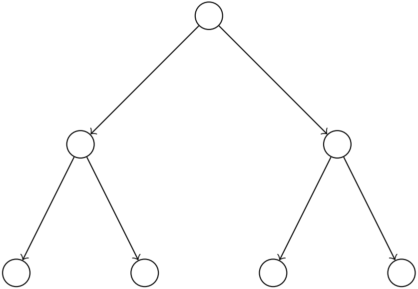
Motivation

AlgoSyn: Algorithmic Synthesis of Reactive and Discrete-Continuous Systems

Part of a pumping station



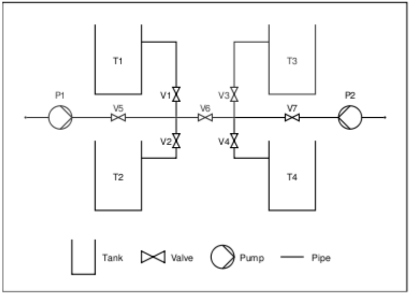
Structures we usually consider



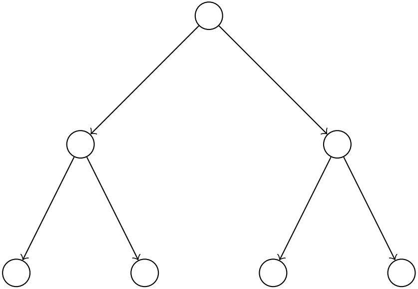
Motivation

AlgoSyn: Algorithmic Synthesis of Reactive and Discrete-Continuous Systems

Part of a pumping station



Structures we usually consider



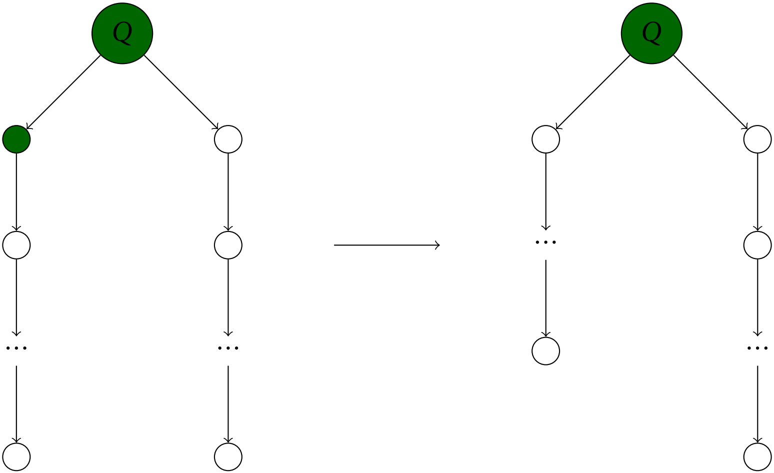
Can we model states by arbitrary relational structures?

(1) change described using appropriate rewriting rules

(2) properties given in MSO on structures + temporal logic for change

Example: Two Counter Machine

Example: decrement first counter



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

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

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

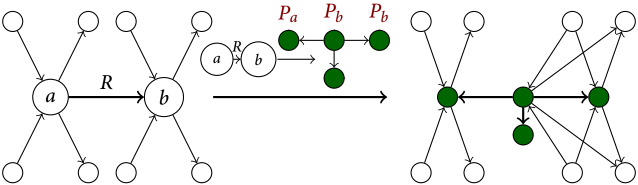
Rewriting Definition

$\mathfrak{B} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}/\sigma]$ 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 $l \in L\} \cup \{(a, a) \mid a \in A\}$,

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

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

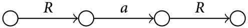
Rewriting Example



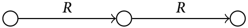
Simple Structure Rewriting

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

Separated:



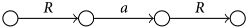
Not Separated:



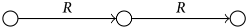
Simple Structure Rewriting

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

Separated:



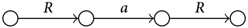
Not Separated:

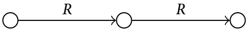


Simple Rule $\mathfrak{L} \to \mathfrak{R}$: $\mathfrak{R}$ is **separated** and $\mathfrak{L}$ is a **single tuple in relation**

Simple Structure Rewriting

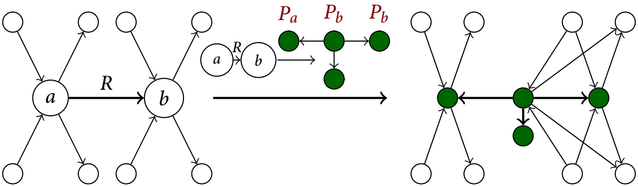
Separated Structures: no element is in two non-terminal relations **(Courcelle, Engelfriet, Rozenberg, 1991)**

Separated: 

Not Separated: 

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

Example



Structure Rewriting Games

Finite game graph with edges labelled by simple rewriting rules.

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

Structure Rewriting Games

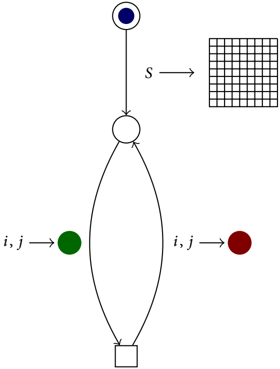
Finite game graph with edges labelled by simple rewriting rules.

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

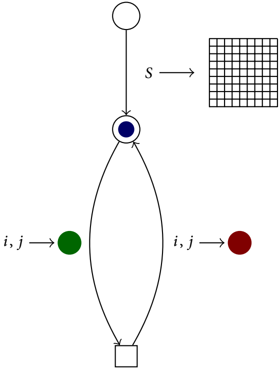
Why Universal Rewriting for Games?

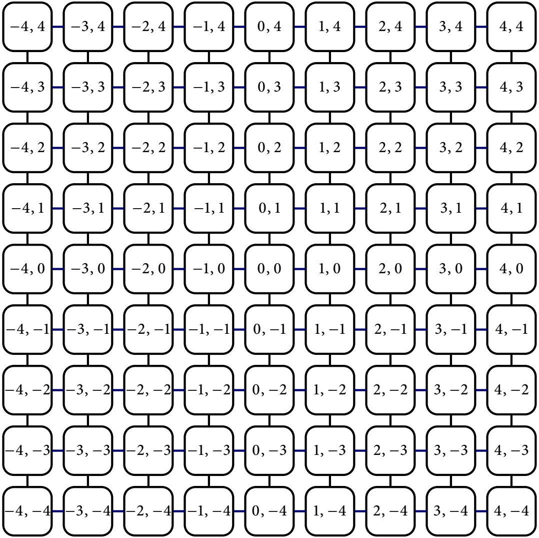
- **In contrast** to **graph grammars** (single player)
- Establishing the winner if players **pick** embeddings is **undecidable:**
  - simulate **active context-free games** (thanks to **Anca Muscholl**)
- Choosing embedding can be allowed in special cases
  - e.g. for a **bounded number of non-terminals**

Example: Game Played with Structures (Gomoku)

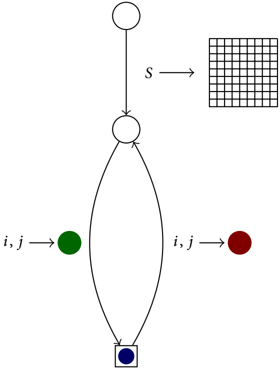


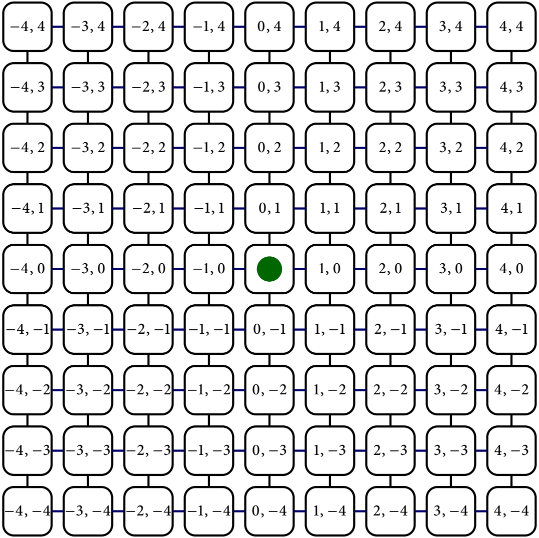
Example: Game Played with Structures (Gomoku)



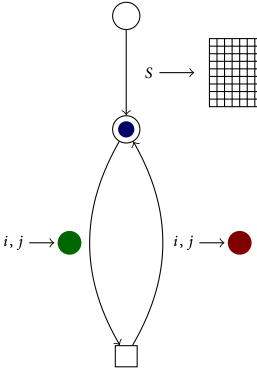


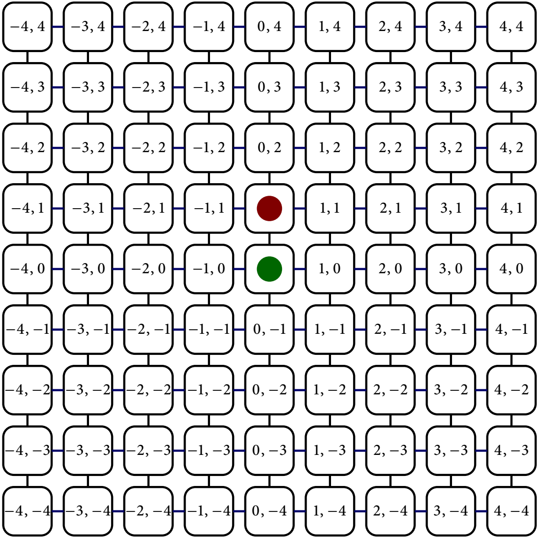
Example: Game Played with Structures (Gomoku)



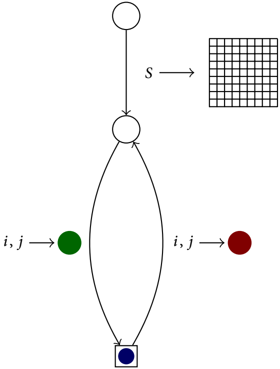


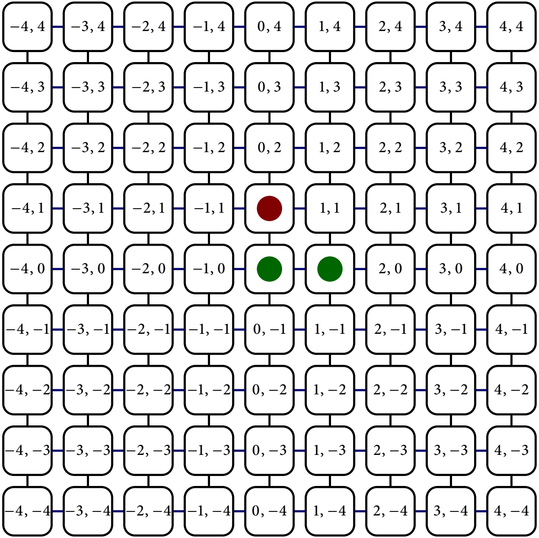
Example: Game Played with Structures (Gomoku)



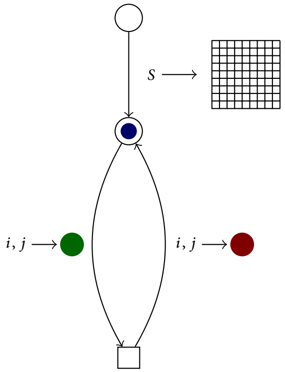


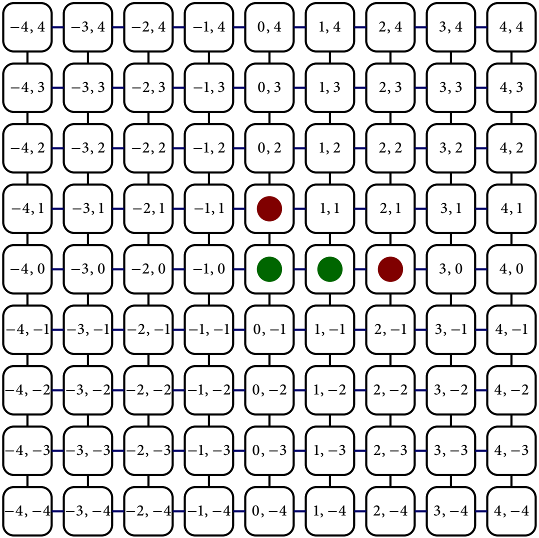
Example: Game Played with Structures (Gomoku)





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,** or in LTL
- **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 $\{\pi \in R^\omega \ : \ (\lim)S(\pi) \vDash \varphi\}$ is **ω-regular**.*

Corollary

Establishing the winner of finite separated rewriting games is decidable.

Intuition: 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:

Intuition: 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:

Intuition: 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:

Intuition: 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:

$$\bullet \longrightarrow \bullet$$

Intuition: 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:

$$\bullet \longrightarrow \bullet \qquad \bullet$$

Intuition: 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:

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

Intuition: 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:

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

Intuition: 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:

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

Intuition: 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:

$$\bullet \longrightarrow \bullet \longrightarrow \bullet \qquad \bullet$$

Intuition: 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:

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

Intuition: 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:

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

Intuition: 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:

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

Intuition: 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:

$$\begin{tikzcd}
\bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] \arrow[ll, bend right] & \bullet \arrow[r] \arrow[lll, bend right] & \cdots \arrow[r] & \bullet \arrow[lllll, bend right]
\end{tikzcd}$$

Proof: Interpreting a Graph in a Tree

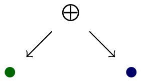
Description of how to build $\mathcal{G}$ is a tree $\mathcal{T}(\mathcal{G})$:

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

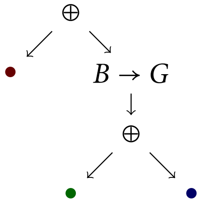
Description of how to build $\mathcal{G}$ is a tree $\mathcal{T}(\mathcal{G})$:

$$\bullet \longrightarrow \bullet$$

$$\begin{tikzcd}
 & B \to G \arrow[d] & \\
 & \oplus \arrow[dl] \arrow[dr] & \\
\bullet & & \bullet
\end{tikzcd}$$

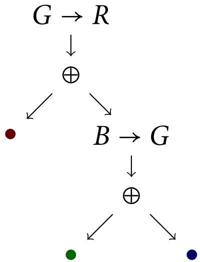
Proof: Interpreting a Graph in a Tree

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



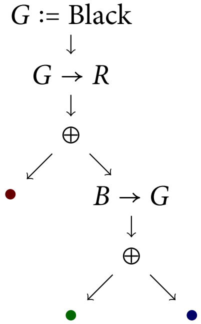
Proof: Interpreting a Graph in a Tree

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



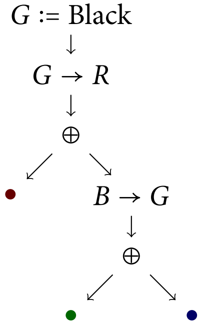
Proof: Interpreting a Graph in a Tree

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

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

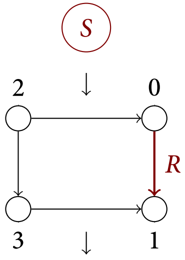
Proof: Separated Rewriting as a Tree

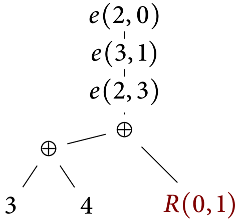
$$S$$

$$\downarrow$$

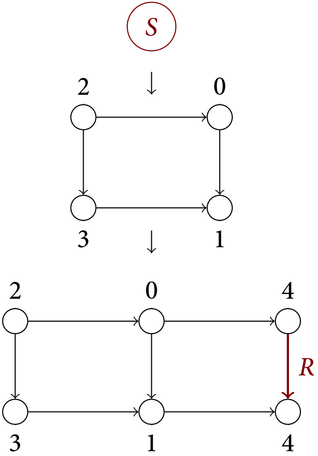
$$S$$

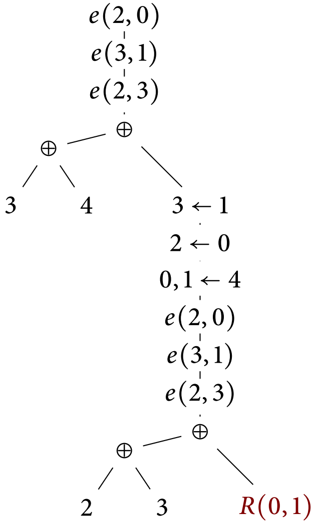
Proof: Separated Rewriting as a Tree



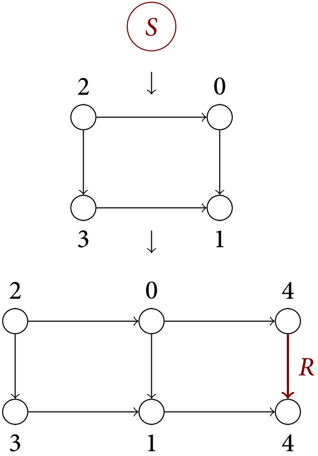


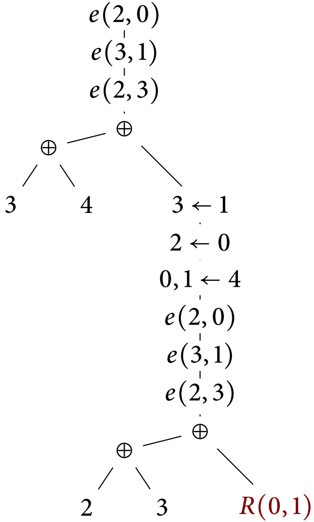
Proof: Separated Rewriting as a Tree





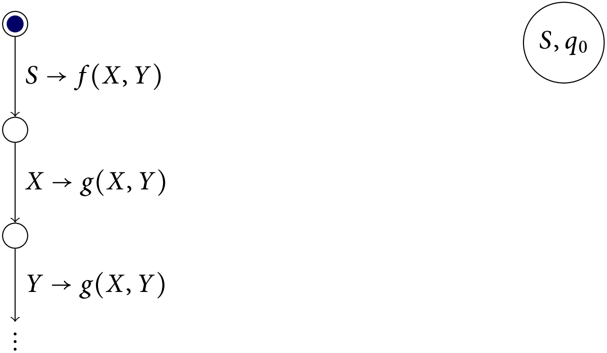
Proof: Separated Rewriting as a Tree



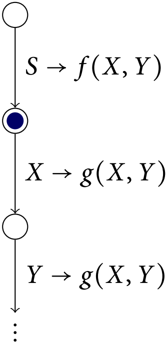


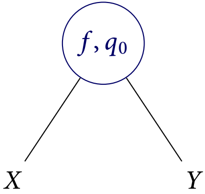
MSO-to-MSO interpretation: $\varphi \to \psi$

Proof: From Tree to Alternating Word Automata

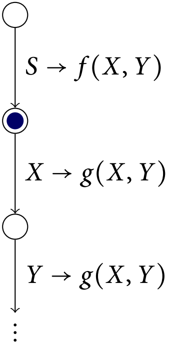


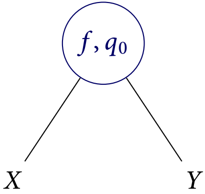
Proof: From Tree to Alternating Word Automata





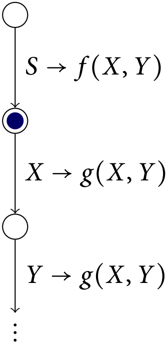
Proof: From Tree to Alternating Word Automata

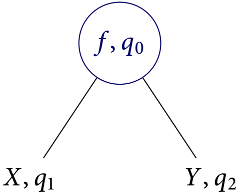




existential: pick transition

Proof: From Tree to Alternating Word Automata

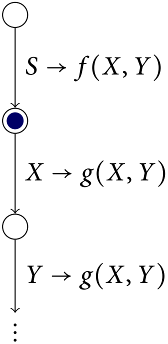


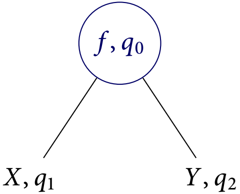


existential: pick transition

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

Proof: From Tree to Alternating Word Automata



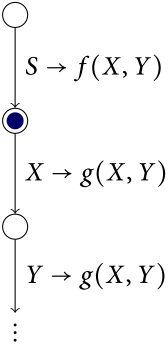


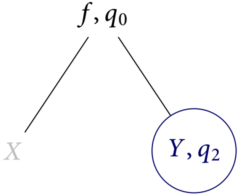
existential: pick transition

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

universal: left or right

Proof: From Tree to Alternating Word Automata



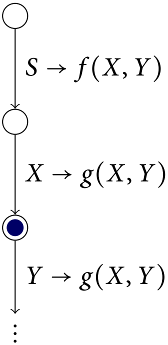


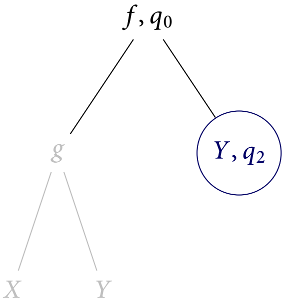
existential: pick transition

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

universal: left or right

Proof: From Tree to Alternating Word Automata





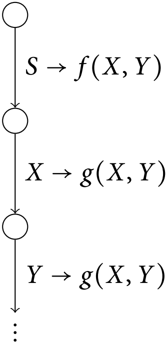
existential: pick transition

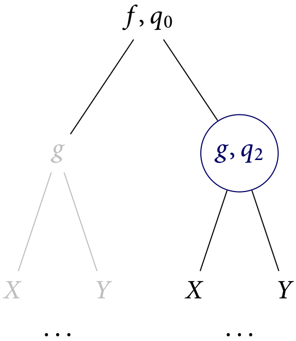
universal: left or right

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

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

Basic Extensions

- The way of **combining sides of a rule** can be generalised
- The theorem on **separated games can be generalised:**
  - to anything known about **ω-regular games**
  - to **some infinite arenas** e.g. pushdown graphs

Outlook

Basic Extensions

- The way of **combining sides of a rule** can be generalised
- The theorem on **separated games can be generalised:**
  - to anything known about **ω-regular games**
  - to **some infinite arenas** e.g. pushdown graphs

Further Questions

- **Unary predicates left and right: Petri Nets**, generalisations?
- Other **logics and corresponding graph measures**, e.g. FO, FO[Reach]**?**
- Apply **higher-order** recursion schemes, **hierarchical structures?**
- Can we add **continuous dynamics?**
  - e.g. using $\mathbb{R}$**-structures** or timed automata
  - simple **quantitative logics** can be used
- Can we use **abstraction** for more complex rewriting systems?

Outlook

Basic Extensions

- The way of **combining sides of a rule** can be generalised
- The theorem on **separated games can be generalised:**
  - to anything known about **ω-regular games**
  - to **some infinite arenas** e.g. pushdown graphs

Further Questions

- **Unary predicates left and right: Petri Nets**, generalisations?
- Other **logics and corresponding graph measures**, e.g. FO, FO[Reach]**?**
- Apply **higher-order** recursion schemes, **hierarchical structures?**
- Can we add **continuous dynamics?**
  - e.g. using **$\mathbb{R}$-structures** or timed automata
  - simple **quantitative logics** can be used
- Can we use **abstraction** for more complex rewriting systems?

Thank You