

The transformational approach

Gabriele Puppis

LaBRI / CNRS



MSO-interpretation as a graph transformation



- 1 Start from a graph G (e.g. the binary tree) that has a decidable MSO-theory
- 2 Transform G into a new graph G' by logically defining nodes and edges of G' inside G
- 3 Given any MSO property ψ over G' , decide it by rephrasing it into a property over G

Definition

An **MSO-interpretation** is a tuple \mathcal{I} of formulas

$$\underbrace{\varphi_{\text{dom}}(x)}_{\text{domain formula}} \quad \underbrace{\varphi_{e_1}(x, y) \dots \varphi_{e_k}(x, y)}_{\text{edge formulas}} \quad \underbrace{\varphi_{a_1}(x) \dots \varphi_{a_m}(x)}_{\text{color formulas}}$$

defining a transformation from graph G to graph $\mathcal{I}(G)$ such that

- v is a vertex of $\mathcal{I}(G)$ iff $(G, v) \models \varphi_{\text{dom}}(x)$
- (u, v) is an e -edge of $\mathcal{I}(G)$ iff $(G, u, v) \models \varphi_e(x, y)$
- v has color a in $\mathcal{I}(G)$ iff $(G, v) \models \varphi_a(x)$

Definition

An **MSO-interpretation** is a tuple \mathcal{I} of formulas

$$\underbrace{\varphi_{\text{dom}}(x)}_{\text{domain formula}} \quad \underbrace{\varphi_{e_1}(x, y) \dots \varphi_{e_k}(x, y)}_{\text{edge formulas}} \quad \underbrace{\varphi_{a_1}(x) \dots \varphi_{a_m}(x)}_{\text{color formulas}}$$

defining a transformation from graph G to graph $\mathcal{I}(G)$ such that

- v is a vertex of $\mathcal{I}(G)$ iff $(G, v) \models \varphi_{\text{dom}}(x)$
- (u, v) is an e -edge of $\mathcal{I}(G)$ iff $(G, u, v) \models \varphi_e(x, y)$
- v has color a in $\mathcal{I}(G)$ iff $(G, v) \models \varphi_a(x)$

Definition

An **MSO-interpretation** is a tuple \mathcal{I} of formulas

$$\underbrace{\varphi_{\text{dom}}(x)}_{\text{domain formula}} \quad \underbrace{\varphi_{e_1}(x, y) \dots \varphi_{e_k}(x, y)}_{\text{edge formulas}} \quad \underbrace{\varphi_{a_1}(x) \dots \varphi_{a_m}(x)}_{\text{color formulas}}$$

defining a transformation from graph G to graph $\mathcal{I}(G)$ such that

- v is a vertex of $\mathcal{I}(G)$ iff $(G, v) \models \varphi_{\text{dom}}(x)$
- (u, v) is an e -edge of $\mathcal{I}(G)$ iff $(G, u, v) \models \varphi_e(x, y)$
- v has color a in $\mathcal{I}(G)$ iff $(G, v) \models \varphi_a(x)$

Definition

An **MSO-interpretation** is a tuple \mathcal{I} of formulas

$$\underbrace{\varphi_{\text{dom}}(x)}_{\text{domain formula}} \quad \underbrace{\varphi_{e_1}(x, y) \dots \varphi_{e_k}(x, y)}_{\text{edge formulas}} \quad \underbrace{\varphi_{a_1}(x) \dots \varphi_{a_m}(x)}_{\text{color formulas}}$$

defining a transformation from graph G to graph $\mathcal{I}(G)$ such that

- v is a vertex of $\mathcal{I}(G)$ iff $(G, v) \models \varphi_{\text{dom}}(x)$
- (u, v) is an e -edge of $\mathcal{I}(G)$ iff $(G, u, v) \models \varphi_e(x, y)$
- v has color a in $\mathcal{I}(G)$ iff $(G, v) \models \varphi_a(x)$

Definition

An **MSO-interpretation** is a tuple \mathcal{I} of formulas

$$\underbrace{\varphi_{\text{dom}}(x)}_{\text{domain formula}} \quad \underbrace{\varphi_{e_1}(x, y) \dots \varphi_{e_k}(x, y)}_{\text{edge formulas}} \quad \underbrace{\varphi_{a_1}(x) \dots \varphi_{a_m}(x)}_{\text{color formulas}}$$

defining a transformation from graph G to graph $\mathcal{I}(G)$ such that

- v is a vertex of $\mathcal{I}(G)$ iff $(G, v) \models \varphi_{\text{dom}}(x)$
- (u, v) is an e -edge of $\mathcal{I}(G)$ iff $(G, u, v) \models \varphi_e(x, y)$
- v has color a in $\mathcal{I}(G)$ iff $(G, v) \models \varphi_a(x)$

Transfer theorem

Every sentence ψ can be transformed into $\mathcal{I}^{-1}(\psi)$ such that

$$\mathcal{I}(G) \models \psi \quad \text{iff} \quad G \models \mathcal{I}^{-1}(\psi)$$

Hence, if G has decidable MSO theory, then so has $\mathcal{I}(G)$.

Definition

An **MSO-interpretation transduction** is a tuple \mathcal{I} of formulas

$$\underbrace{\varphi_{\text{dom}}^i(x)}_{\text{domain formula}} \quad \underbrace{\varphi_{e_1}^{i,j}(x,y) \dots \varphi_{e_k}^{i,j}(x,y)}_{\text{edge formulas}} \quad \underbrace{\varphi_{a_1}^i(x) \dots \varphi_{a_m}^i(x)}_{\text{color formulas}}$$

defining a transformation from graph G to graph $\mathcal{I}(G)$ such that

- (i, v) is a vertex of $\mathcal{I}(G)$ iff $(G, v) \models \varphi_{\text{dom}}^i(x)$
- $((i, u), (j, v))$ is an e -edge of $\mathcal{I}(G)$ iff $(G, u, v) \models \varphi_e^{i,j}(x, y)$
- (i, v) has color a in $\mathcal{I}(G)$ iff $(G, v) \models \varphi_a^i(x)$

Transfer theorem

Every sentence ψ can be transformed into $\mathcal{I}^{-1}(\psi)$ such that

$$\mathcal{I}(G) \models \psi \quad \text{iff} \quad G \models \mathcal{I}^{-1}(\psi)$$

Hence, if G has decidable MSO theory, then so has $\mathcal{I}(G)$.

Most edge formulas can be abbreviated by **regular expressions**:

- “ a ” abbreviates $(x, y) \in E_a$
- “ \bar{a} ” abbreviates $(y, x) \in E_a$
- “ $a + b$ ” abbreviates $(x, y) \in E_a \vee (x, y) \in E_b$
- “ $a \cdot b$ ” abbreviates $\exists z. (x, z) \in E_a \wedge (z, y) \in E_b$
- “ a^* ” abbreviates $(x, y) \in E_a^*$
- “ c ” abbreviates $c(x)$ (assume vertex colors \neq edge labels)

Most edge formulas can be abbreviated by **regular expressions**:

- “ a ” abbreviates $(x, y) \in E_a$
- “ \bar{a} ” abbreviates $(y, x) \in E_a$
- “ $a + b$ ” abbreviates $(x, y) \in E_a \vee (x, y) \in E_b$
- “ $a \cdot b$ ” abbreviates $\exists z. (x, z) \in E_a \wedge (z, y) \in E_b$
- “ a^* ” abbreviates $(x, y) \in E_a^*$
- “ c ” abbreviates $c(x)$ (assume vertex colors \neq edge labels)

Example

The regular expression “ $a^* \cdot (c + d) \cdot \bar{b}$ ” describes a formula $\varphi(x, y)$ that witnesses a path from x to y such that

- 1 traverses a sequence of a -labelled edges
- 2 reaches a vertex with color c or d
- 3 finally traverses in backward direction a b -labelled edge

Most edge formulas can be abbreviated by **regular expressions**:

- “ a ” abbreviates $(x, y) \in E_a$
- “ \bar{a} ” abbreviates $(y, x) \in E_a$
- “ $a + b$ ” abbreviates $(x, y) \in E_a \vee (x, y) \in E_b$
- “ $a \cdot b$ ” abbreviates $\exists z. (x, z) \in E_a \wedge (z, y) \in E_b$
- “ a^* ” abbreviates $(x, y) \in E_a^*$
- “ c ” abbreviates $c(x)$ (assume vertex colors \neq edge labels)



Example

The regular expression “ $a^* \cdot (c + d) \cdot \bar{b}$ ” describes a formula $\varphi(x, y)$ that witnesses a path from x to y such that

- 1 traverses a sequence of a -labelled edges
- 2 reaches a vertex with color c or d
- 3 finally traverses in backward direction a b -labelled edge

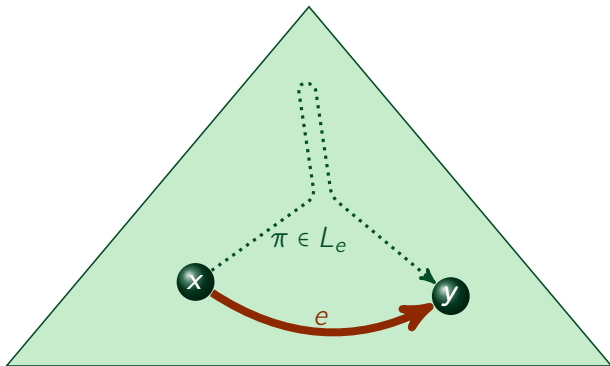
Special forms of interpretations on trees

A **rational restriction** is an MSO-interpretation of tree defined by

$$\varphi_{\text{dom}}(x) = \text{“}\exists \text{ path } \pi \text{ from root to } x \text{ such that } \pi \in L\text{”}$$

Similarly, an **inverse rational mapping** is defined as

$$\varphi_e(x, y) = \text{“}\exists \text{ path } \pi \text{ from } x \text{ to } y \text{ such that } \pi \in L_e\text{”}$$



Definition

A **pushdown system** is a tuple $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta)$, where

- Q is a finite set of control states
- Σ is a finite alphabet for transition labels
- Γ is a finite alphabet for stack symbols
- $\Delta \subseteq Q \times \Gamma \times \Sigma \times Q \times \Gamma^*$ is a finite set of transition rules

Definition

A **pushdown system** is a tuple $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta)$, where

- Q is a finite set of control states
- Σ is a finite alphabet for transition labels
- Γ is a finite alphabet for stack symbols
- $\Delta \subseteq Q \times \Gamma \times \Sigma \times Q \times \Gamma^*$ is a finite set of transition rules

Configurations = pairs $(\underbrace{q}_{\text{state}}, \underbrace{w}_{\text{stack}}) \in Q \times \Gamma^*$

Transitions = $(q, \gamma w) \xrightarrow{a} (q', v w)$
iff $(q, \gamma, a, q', v) \in \Delta$

Definition

A **pushdown system** is a tuple $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta)$, where

- Q is a finite set of control states
- Σ is a finite alphabet for transition labels
- Γ is a finite alphabet for stack symbols
- $\Delta \subseteq Q \times \Gamma \times \Sigma \times Q \times \Gamma^*$ is a finite set of transition rules

Configurations = pairs $(\underbrace{q}_{\text{state}}, \underbrace{w}_{\text{stack}}) \in Q \times \Gamma^*$

Transitions = $(q, \gamma w) \xrightarrow{a} (q', v w)$
iff $(q, \gamma, a, q', \underbrace{v}_{\in \Gamma^{\leq 2}}) \in \Delta$

 W.l.o.g. assume that stack length changes at most by 1

Interest in properties of transition graphs of pushdown systems

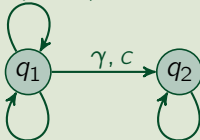
Context-free graph

A **connected component** of a graph is a maximal subgraph in which every two vertices can be connected by a path that traverses edges in either direction.

A **context-free graph** is a connected component of the transition graph of a pushdown system.

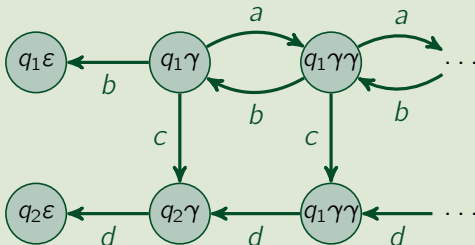
Example

$\gamma, a : \text{push } \gamma$



$\gamma, b : \text{pop } \gamma$

$\gamma, d : \text{pop } \gamma$



Theorem (Caucal '96)

Context-free graphs are definable in the binary tree using rational restrictions and inverse finite mappings.

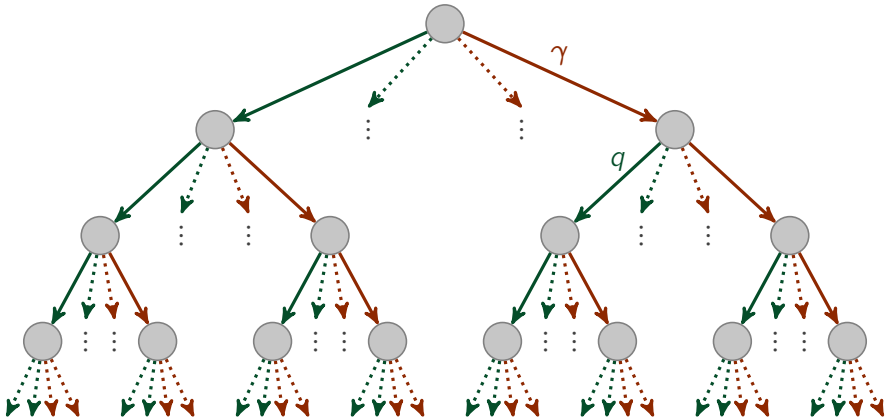
Proof sketch in the next slide...

Corollary (Muller & Schupp '85)

MSO is decidable over context-free graphs.

Context-free graphs are definable by restrictions and inverse mappings.

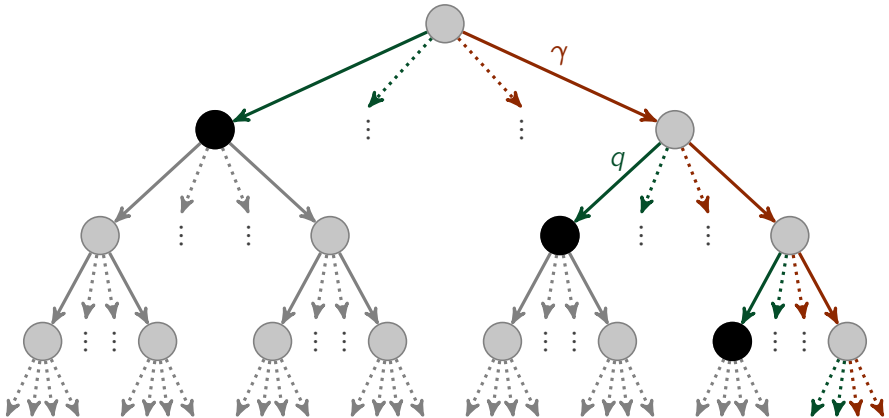
Consider a pushdown system $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta)$ and the $Q \uplus \Gamma$ -labelled tree (interpretable in the binary tree)



Context-free graphs are definable by restrictions and inverse mappings.

Consider a pushdown system $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta)$ and the $Q \uplus \Gamma$ -labelled tree (interpretable in the binary tree)

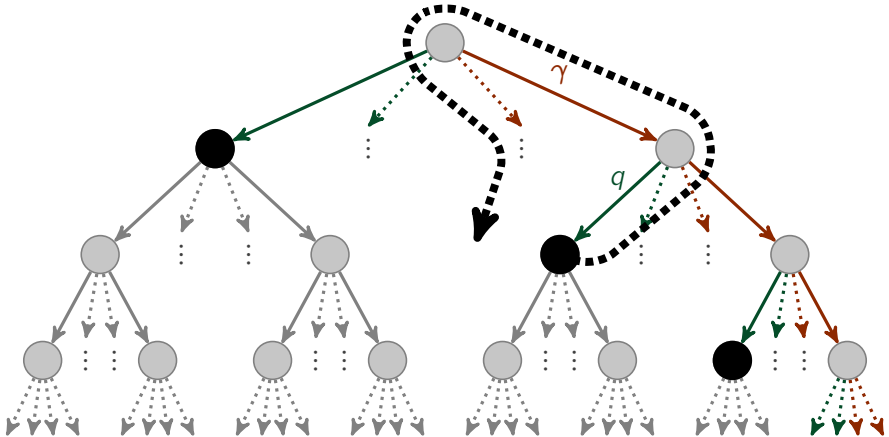
- configurations of \mathcal{P} : $\varphi_{\text{dom}}(x) = \exists \pi_{\text{root},x} \in \Gamma^* \cdot Q$



Context-free graphs are definable by restrictions and inverse mappings.

Consider a pushdown system $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta)$ and the $Q \uplus \Gamma$ -labelled tree (interpretable in the binary tree)

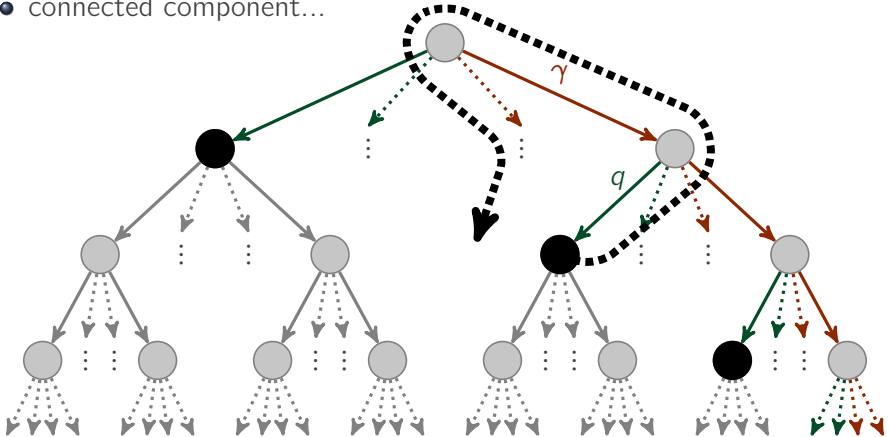
- configurations of \mathcal{P} : $\varphi_{\text{dom}}(x) = \exists \pi_{\text{root},x} \in \Gamma^* \cdot Q$
- transitions of \mathcal{P} : $\varphi_a(x, y) = \exists \pi_{x,y} \in \bigcup_{(q,\gamma,a,q',v) \in \Delta} (\bar{q} \cdot \bar{\gamma} \cdot v^{\text{rev}} \cdot q')$



Context-free graphs are definable by restrictions and inverse mappings.

Consider a pushdown system $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta)$ and the $Q \uplus \Gamma$ -labelled tree (interpretable in the binary tree)

- configurations of \mathcal{P} : $\varphi_{\text{dom}}(x) = \exists \pi_{\text{root},x} \in \Gamma^* \cdot Q$
- transitions of \mathcal{P} : $\varphi_a(x, y) = \exists \pi_{x,y} \in \bigcup_{(q,\gamma,a,q',v) \in \Delta} (\bar{q} \cdot \bar{\gamma} \cdot v^{\text{rev}} \cdot q')$
- connected component...



The previous result can be lifted to prefix-rewriting systems, which generalize pushdown systems by giving graphs with infinite degree:

- no distinction between control states and stack letters
(a single alphabet is used)
- less restricted forms of rewriting rules
(more than one letter can be rewritten in a single transition)

The previous result can be lifted to prefix-rewriting systems, which generalize pushdown systems by giving graphs with infinite degree:

- no distinction between control states and stack letters
(a single alphabet is used)
- less restricted forms of rewriting rules
(more than one letter can be rewritten in a single transition)

Definition

A **prefix-rewriting system** is a tuple $\mathcal{P} = (\Sigma, \Gamma, \Delta)$, where

- Σ is a finite alphabet for transition labels
- Γ is a finite alphabet for “stack” symbols
- Δ is a finite set of transition rules of the form (U, a, V) , with $a \in \Sigma$ and U, V **regular languages over Γ** .

The previous result can be lifted to prefix-rewriting systems, which generalize pushdown systems by giving graphs with infinite degree:

- no distinction between control states and stack letters
(a single alphabet is used)
- less restricted forms of rewriting rules
(more than one letter can be rewritten in a single transition)

Definition

A **prefix-rewriting system** is a tuple $\mathcal{P} = (\Sigma, \Gamma, \Delta)$, where

- Σ is a finite alphabet for transition labels
- Γ is a finite alphabet for “stack” symbols
- Δ is a finite set of transition rules of the form (U, a, V) , with $a \in \Sigma$ and U, V **regular languages over Γ** .

Configurations = words in Γ^* (or in some regular language)

Transitions = $u w \xrightarrow{a} v w$

iff $u \in U$ and $v \in V$ for some $(U, a, V) \in \Delta$

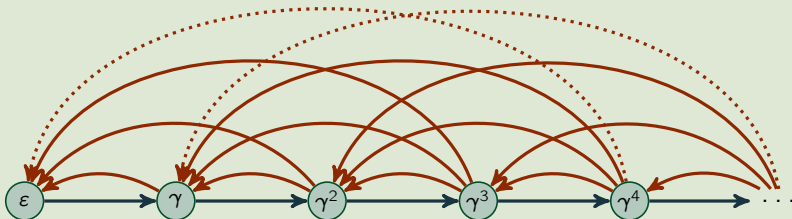
Definition

A **prefix-recognizable graph** is the transition graph of a prefix-rewriting system.

Example

Consider the prefix rewriting system $\mathcal{P} = (\Sigma, \Gamma, \Delta)$, where

- $\Sigma = \{\text{succ}, \text{smaller}\}$
- $\Gamma = \{\gamma\}$
- Δ consists of the two rules $(\{\varepsilon\}, \text{succ}, \{\gamma\})$ and $(\{\gamma\}^+, \text{smaller}, \{\varepsilon\})$



Theorem (Caucal '96)

Prefix-recognizable graphs are definable in the binary tree using rational restrictions and inverse rational mappings.

Exactly the same proof as before...

$$u w \xrightarrow{a} v w$$

iff

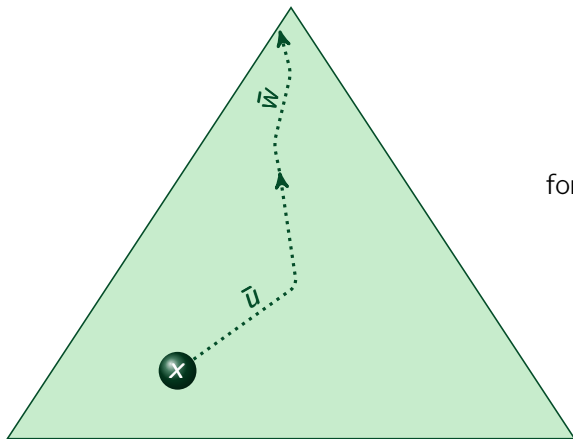
$$u \in U \text{ and } v \in V$$

for some $(U, a, V) \in \Delta$

Theorem (Caucal '96)

Prefix-recognizable graphs are definable in the binary tree using rational restrictions and inverse rational mappings.

Exactly the same proof as before...



$$u w \xrightarrow{a} v w$$

iff

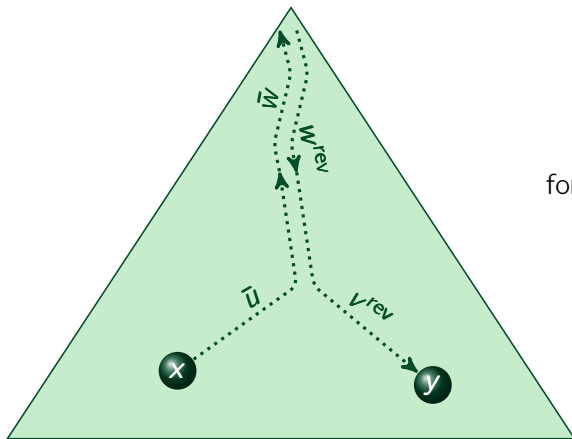
$$u \in U \text{ and } v \in V$$

for some $(U, a, V) \in \Delta$

Theorem (Caucal '96)

Prefix-recognizable graphs are definable in the binary tree using rational restrictions and inverse rational mappings.

Exactly the same proof as before...



$$u w \xrightarrow{a} v w$$

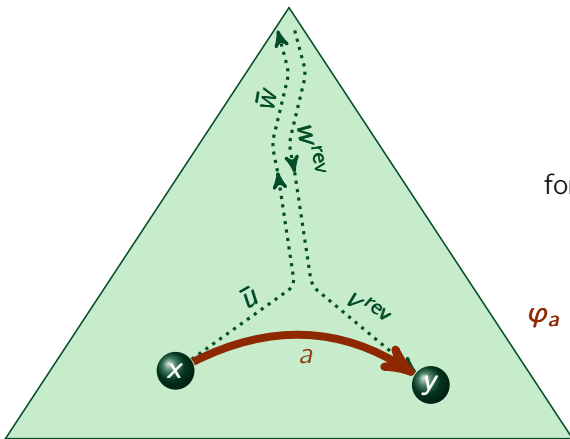
iff

$u \in U$ and $v \in V$
for some $(U, a, V) \in \Delta$

Theorem (Caucal '96)

Prefix-recognizable graphs are definable in the binary tree using rational restrictions and inverse rational mappings.

Exactly the same proof as before...



$$u w \xrightarrow{a} v w$$

iff

$$u \in U \text{ and } v \in V$$

for some $(U, a, V) \in \Delta$

$$\varphi_a = \bigcup_{(U, a, V) \in \Delta} \bar{U} \cdot V^{\text{rev}}$$

We just saw that

- **context-free graphs** can be defined in the binary tree using *rational restrictions* and *inverse finite mappings*
- **prefix-recognizable graphs** can be defined in the binary tree using *rational restrictions* and *inverse rational mappings*

We just saw that

- **context-free graphs** can be defined in the binary tree using *rational restrictions* and *inverse finite mappings*
- **prefix-recognizable graphs** can be defined in the binary tree using *rational restrictions* and *inverse rational mappings*

The converse is also true:

Theorem (Caucal '96)

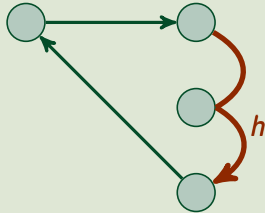
The graphs that can be defined in the binary tree using rational restrictions and inverse finite / rational mappings are context-free / prefix-recognizable.

Context-free graphs have alternative representations based on **hyperedge-replacement**

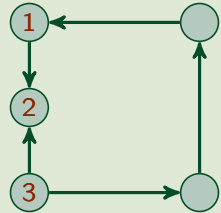
Context-free graphs have alternative representations based on **hyperedge-replacement**

Example of hyperedge replacement

G :

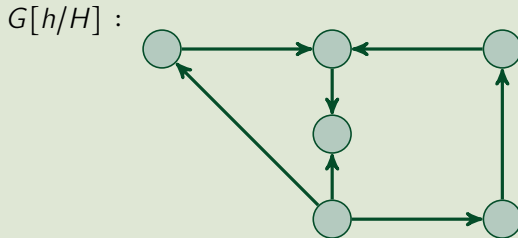
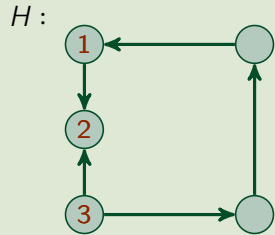
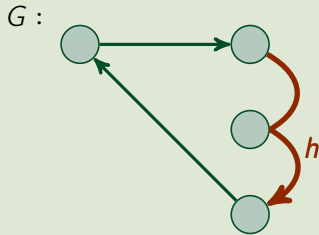


H :



Context-free graphs have alternative representations based on **hyperedge-replacement**

Example of hyperedge replacement



Some definitions

A **hyperedge** is a sequence of vertices $h = (v_1, \dots, v_k)$

Some definitions

A **hyperedge** is a sequence of vertices $h = (v_1, \dots, v_k)$

A **hypergraph** is a structure of vertices and hyperedges (different hyperedges are given different labels).

Some definitions

A **hyperedge** is a sequence of vertices $h = (v_1, \dots, v_k)$

A **hypergraph** is a structure of vertices and hyperedges (different hyperedges are given different labels).

A **hyperedge replacement** is the replacement of a hyperedge $h = (v_1, \dots, v_k)$ in a hypergraph G with another hypergraph H (glueing points are represented by marking vertices of H)

Some definitions

A **hyperedge** is a sequence of vertices $h = (v_1, \dots, v_k)$

A **hypergraph** is a structure of vertices and hyperedges (different hyperedges are given different labels).

A **hyperedge replacement** is the replacement of a hyperedge $h = (v_1, \dots, v_k)$ in a hypergraph G with another hypergraph H (glueing points are represented by marking vertices of H)

A **hyperedge replacement grammar** is a finite set of rewriting rules of the form

$$h_1 \mapsto H_1 \quad \dots \quad h_n \mapsto H_n$$

together with an initial hyperedge h_1 .

Some definitions

A **hyperedge** is a sequence of vertices $h = (v_1, \dots, v_k)$

A **hypergraph** is a structure of vertices and hyperedges (different hyperedges are given different labels).

A **hyperedge replacement** is the replacement of a hyperedge $h = (v_1, \dots, v_k)$ in a hypergraph G with another hypergraph H (glueing points are represented by marking vertices of H)


A **hyperedge replacement grammar** is a finite set of rewriting rules of the form

$$h_1 \mapsto H_1 \quad \dots \quad h_n \mapsto H_n$$

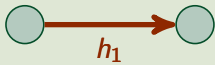
together with an initial hyperedge h_1 .

Finally, one defines the **limit** of a series of replacements

$$h_1 \mapsto H_1 \mapsto H_1[h_{i_2}/H_2] \mapsto \dots$$

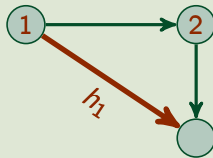
 Since hyperedge replacements are confluent, the limit is the same for all (fair) series of replacements!

Limit graph of a hyperedge replacement grammar

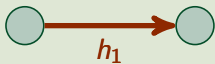


\mapsto

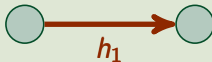
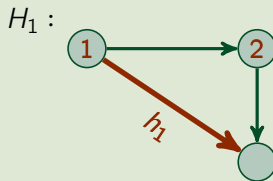
$H_1 :$



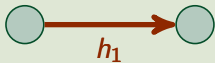
Limit graph of a hyperedge replacement grammar



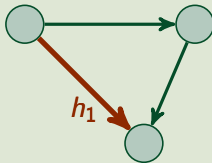
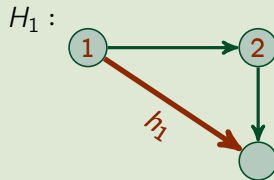
\mapsto



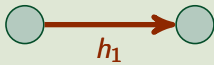
Limit graph of a hyperedge replacement grammar



\mapsto

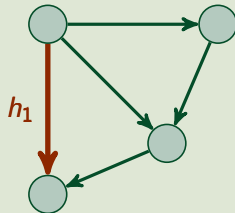
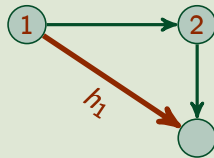


Limit graph of a hyperedge replacement grammar

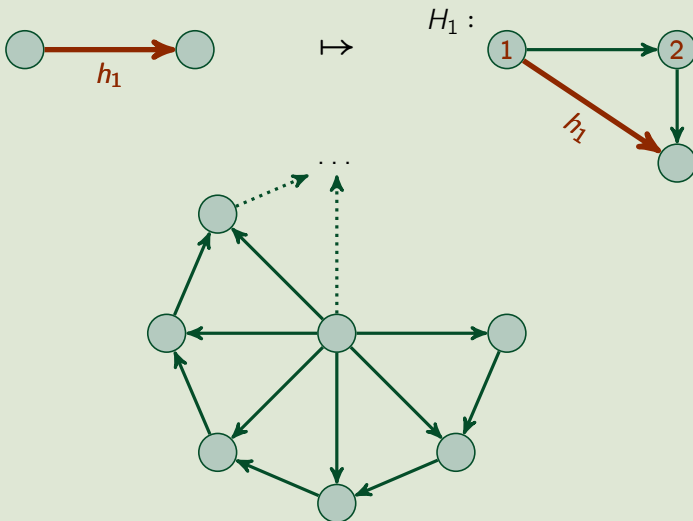


\mapsto

$H_1 :$



Limit graph of a hyperedge replacement grammar



Limit graphs can be **disconnected** and have **unbounded degree**

If we restrict ourselves to **special forms** of grammars, where

- there are no markings on hyperedges
- there are no repetitions of vertices in hyperedges
- vertices of hyperedges have incident terminal edges
- ...

Then:

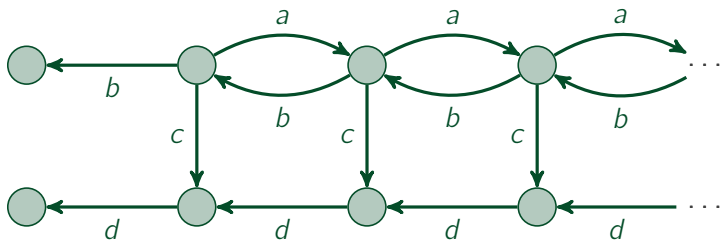
Theorem (Muller & Schupp '85)

The context-free graphs are the limit graphs of special forms of hyperedge-replacement graph grammars.



Proof idea in one direction: consider **end-components**

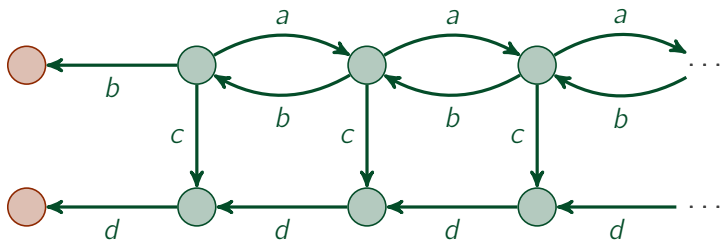
$$V_n = \{ v \mid \text{dist}(v, V_0) \geq n \}$$





Proof idea in one direction: consider **end-components**

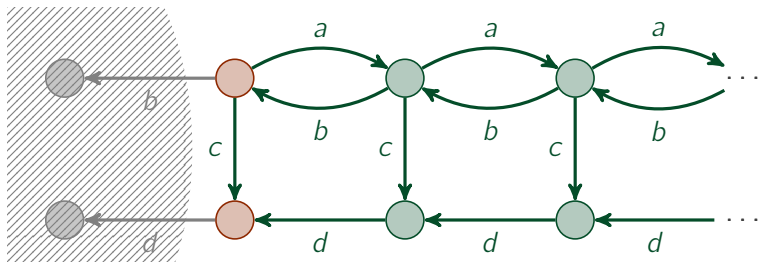
$$V_n = \{ v \mid \text{dist}(v, V_0) \geq n \}$$





Proof idea in one direction: consider **end-components**

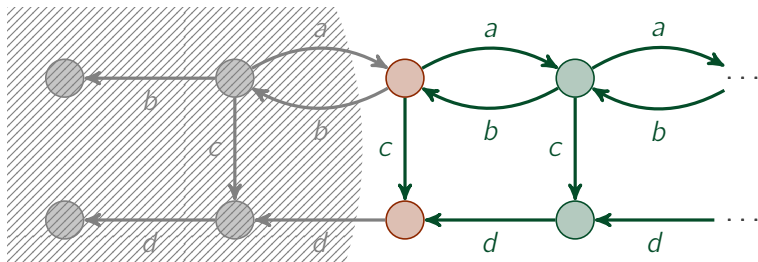
$$V_n = \{ v \mid \text{dist}(v, V_0) \geq n \}$$





Proof idea in one direction: consider **end-components**

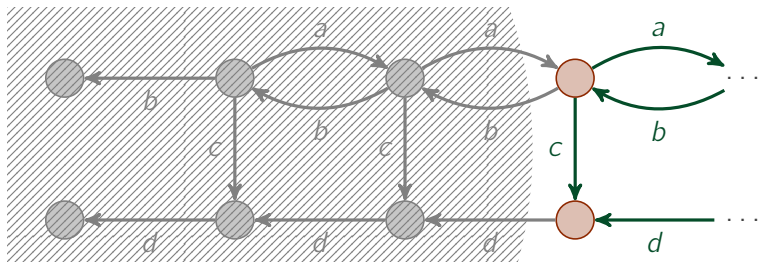
$$V_n = \{ v \mid \text{dist}(v, V_0) \geq n \}$$





Proof idea in one direction: consider **end-components**

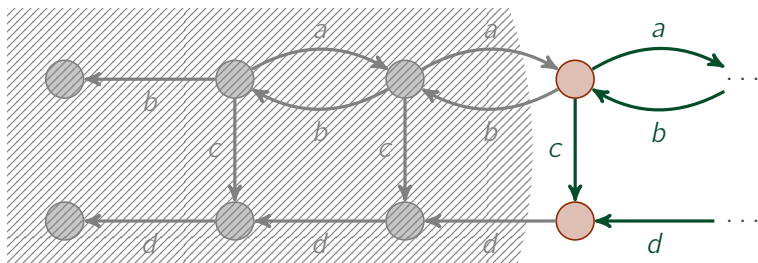
$$V_n = \{ v \mid \text{dist}(v, V_0) \geq n \}$$



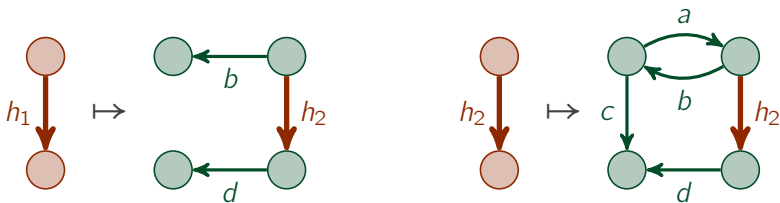


Proof idea in one direction: consider **end-components**

$$V_n = \{ v \mid \text{dist}(v, V_0) \geq n \}$$



Only finitely many non-isomorphic end-components, each one inducing a hyperedge replacement rule:



Analogous results hold for prefix-recognizable graphs:

Theorem (Courcelle '92)

The prefix-recognizable graphs are the limit graphs of **vertex-replacement graph grammars**.

Operations underlying vertex-replacement grammars:

- Disjoint union: $G \uplus G'$
- Vertex relabelling: $G[a/b]$
- Edge creation: $G[a \rightarrow b]$

Transfer theorems can be proved for other transformations besides MSO-interpretations and MSO-transductions, e.g. for unfoldings...

Transfer theorems can be proved for other transformations besides MSO-interpretations and MSO-transductions, e.g. for unfoldings...

Definition

The **unfolding** of a rooted graph G is the tree $\mathbf{unf}(G)$, where

- vertices are the **finite paths** in G originating from the root
- edges are given by **path-extension** relation

i.e. (π, π') is an a -labelled edge in $\mathbf{unf}(G)$ iff

π' is the extension of π with an a -labelled edge in G

Transfer theorems can be proved for other transformations besides MSO-interpretations and MSO-transductions, e.g. for unfoldings...

Definition

The **unfolding** of a rooted graph G is the tree $\text{unf}(G)$, where

- vertices are the **finite paths** in G originating from the root
- edges are given by **path-extension** relation

i.e. (π, π') is an a -labelled edge in $\text{unf}(G)$ iff

π' is the extension of π with an a -labelled edge in G

Transfer theorem (Muchnik '84, Courcelle '96, Walukiewicz '02, ...)

Every sentence ψ can be transformed into $\text{unf}^{-1}(\psi)$ such that

$$\text{unf}(G) \models \psi \quad \text{iff} \quad G \models \text{unf}^{-1}(\psi)$$

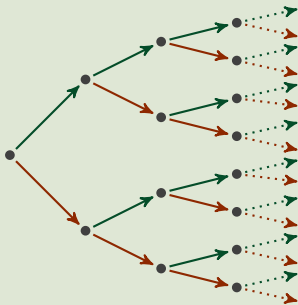


Next slide shows why this subsumes Büchi and Rabin's theorems...

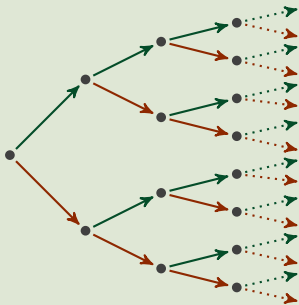
Examples of unfoldings



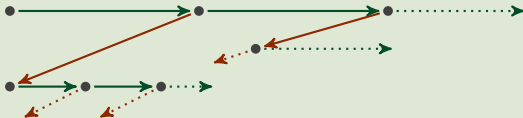
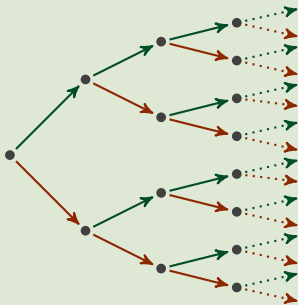
Examples of unfoldings



Examples of unfoldings

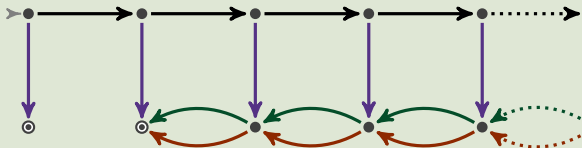


Examples of unfoldings



Application example

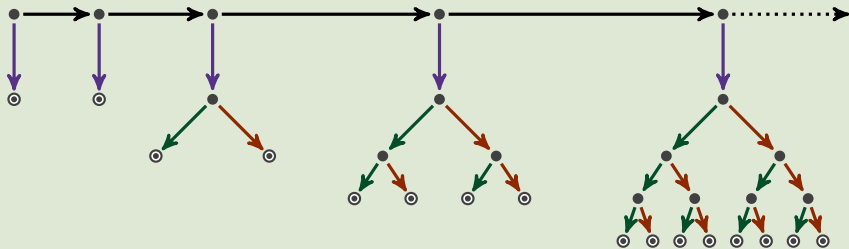
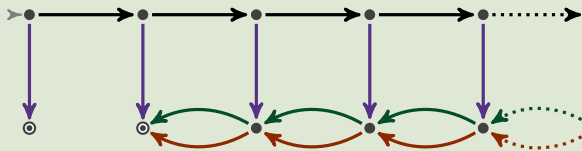
Consider a context-free graph, which has decidable MSO theory.



Application example

Consider a context-free graph, which has decidable MSO theory.

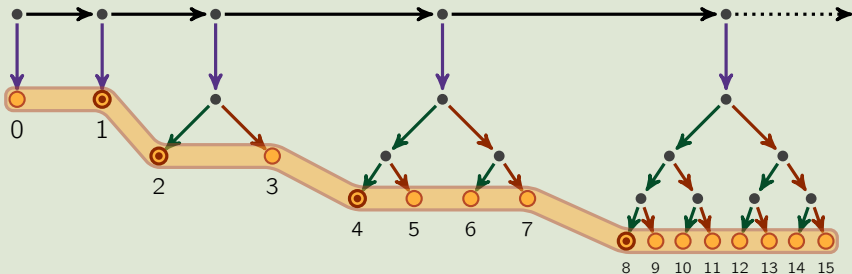
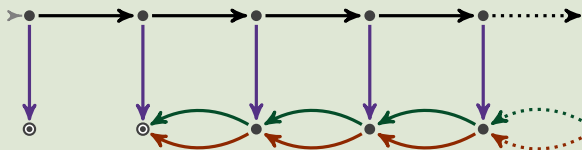
- 1 First apply **unfolding**: this gives a tree with decidable MSO theory



Application example

Consider a context-free graph, which has decidable MSO theory.

- 1 First apply **unfolding**: this gives a tree with decidable MSO theory
- 2 Then apply **MSO-interpretation**: this gives $(\mathbb{N}, +1, \text{Powers})$



Higher-order definable words (e.g. see Fratani & Senizergues '06)

✓ $(\mathbb{N}, +1, \{2^n \mid n \in \mathbb{N}\})$

✓ $(\mathbb{N}, +1, \{2^{2^n} \mid n \in \mathbb{N}\})$

✗ $(\mathbb{N}, +1, \{2^{\cdot^{2^n}} \mid n \in \mathbb{N}\})$

(but MSO is still decidable)

Higher-order definable words (e.g. see Fratani & Senizergues '06)

✓ $(\mathbb{N}, +1, \{2^n \mid n \in \mathbb{N}\})$

✓ $(\mathbb{N}, +1, \{2^{2^n} \mid n \in \mathbb{N}\})$

✗ $(\mathbb{N}, +1, \{2^{\cdot^{\cdot^{\cdot^2}}}\}_{n \text{ times}} \mid n \in \mathbb{N}\})$ (but MSO is still decidable)

✓ $(\mathbb{N}, +1, \{\lfloor n\sqrt{n} \rfloor \mid n \in \mathbb{N}\})$

✓ $(\mathbb{N}, +1, \{\lfloor n \log n \rfloor \mid n \in \mathbb{N}\})$

✓ $(\mathbb{N}, +1, \{n^2 \mid n \in \mathbb{N}\}, \{n^6 \mid n \in \mathbb{N}\}, \{n^{30} \mid n \in \mathbb{N}\}, \dots)$

? $(\mathbb{N}, +1, \{n^2 \mid n \in \mathbb{N}\}, \{n^3 \mid n \in \mathbb{N}\})$ (is MSO decidable?)

Since interpretation and unfolding preserve decidability of MSO theories we can iterate these two operations and produce new graphs...

Definition

The **Caucal hierarchy** is a series of inductively defined graphs and trees:

$$\mathbf{Graphs}_0 = \{ \text{finite graphs} \}$$

$$\mathbf{Trees}_n = \{ \text{unf}(G) \mid G \in \mathbf{Graphs}_n \}$$

$$\mathbf{Graphs}_{n+1} = \{ \mathcal{I}(T) \mid \mathcal{I} \text{ interpretation, } T \in \mathbf{Trees}_n \}$$

Since interpretation and unfolding preserve decidability of MSO theories we can iterate these two operations and produce new graphs...

Definition

The **Caucal hierarchy** is a series of inductively defined graphs and trees:

$$\mathbf{Graphs}_0 = \{ \text{finite graphs} \}$$

$$\mathbf{Trees}_n = \{ \text{unf}(G) \mid G \in \mathbf{Graphs}_n \}$$

$$\mathbf{Graphs}_{n+1} = \{ \mathcal{I}(T) \mid \mathcal{I} \text{ interpretation, } T \in \mathbf{Trees}_n \}$$

Examples

$\mathbf{Trees}_0 = \{ \text{regular trees} \}$, $\mathbf{Graphs}_1 = \{ \text{prefix-recognizable graphs} \}$, ...

Since interpretation and unfolding preserve decidability of MSO theories we can iterate these two operations and produce new graphs...

Definition

The **Caucal hierarchy** is a series of inductively defined graphs and trees:

$$\mathbf{Graphs}_0 = \{ \text{finite graphs} \}$$

$$\mathbf{Trees}_n = \{ \text{unf}(G) \mid G \in \mathbf{Graphs}_n \}$$

$$\mathbf{Graphs}_{n+1} = \{ \mathcal{I}(T) \mid \mathcal{I} \text{ interpretation, } T \in \mathbf{Trees}_n \}$$

Examples

$\mathbf{Trees}_0 = \{ \text{regular trees} \}$, $\mathbf{Graphs}_1 = \{ \text{prefix-recognizable graphs} \}$, ...

Theorem (Caucal '02)

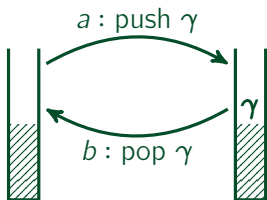
Graphs and trees of Caucal hierarchy have decidable MSO theories.

Theorem (Carayol & Wöhrle '03)

The graphs in **level n of Caucal hierarchy** are ε -closures of transition graphs of **order- n pushdown systems**.

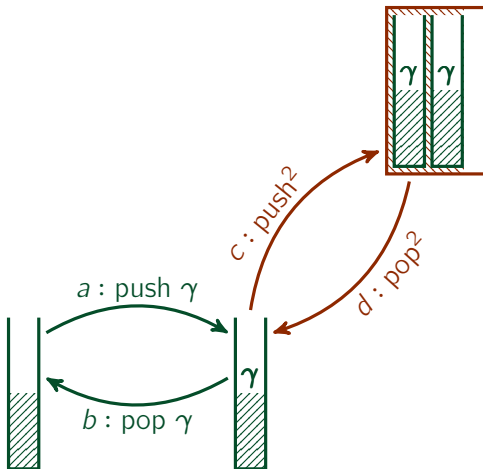
Theorem (Carayol & Wöhrle '03)

The graphs in **level n of Caucal hierarchy** are ε -closures of transition graphs of **order- n pushdown systems**.



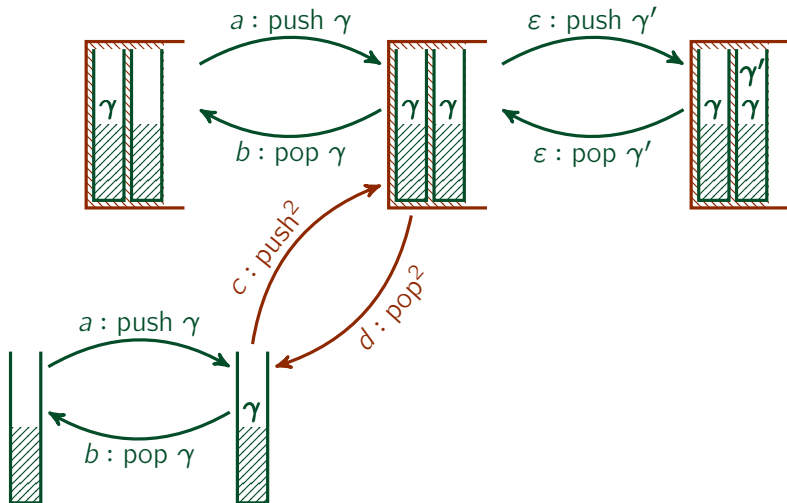
Theorem (Carayol & Wöhrle '03)

The graphs in **level n** of **Caucal hierarchy** are ε -closures of transition graphs of **order- n pushdown systems**.



Theorem (Carayol & Wöhrle '03)

The graphs in **level n of Caucal hierarchy** are ε -closures of transition graphs of **order- n pushdown systems**.



Another example of application of unfolding

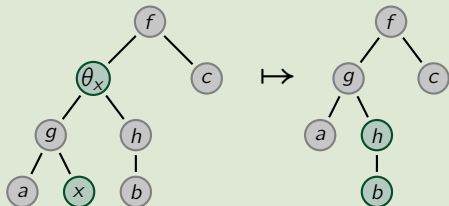
Consider the operation of **substitution of variables by terms**:

$$\theta_x(g, h) \mapsto g[x/h]$$

Another example of application of unfolding

Consider the operation of **substitution of variables by terms**:

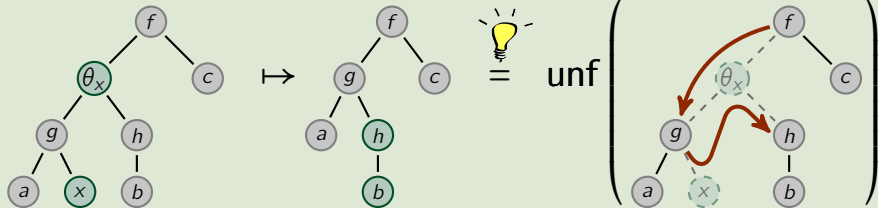
$$\theta_x(g, h) \mapsto g[x/h]$$



Another example of application of unfolding

Consider the operation of **substitution of variables by terms**:

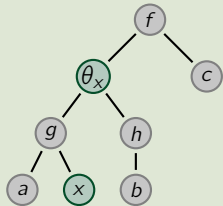
$$\theta_x(g, h) \mapsto g[x/h]$$



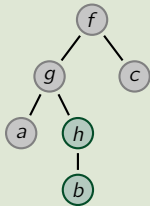
Another example of application of unfolding

Consider the operation of **substitution of variables**:

$$\theta_x(g, h) \mapsto g[x/h]$$

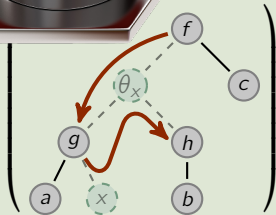


\mapsto



$=$

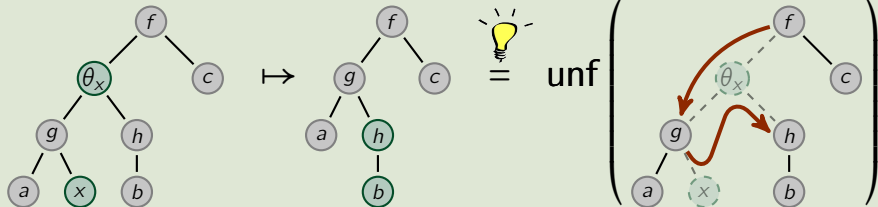
unf



Another example of application of unfolding

Consider the operation of **substitution of variables by terms**:

$$\theta_x(g, h) \mapsto g[x/h]$$



Theorem (Courcelle & Knapik '02)

The operation of substitution of (a fixed number of) variables by terms preserves decidability of MSO theories.

It is convenient to see substitution as a form of **β -reduction in λ -calculus**, but without variable renaming:

$$(\lambda x. g(a, x)) @ h(b) \mapsto g(a, x) [x/h(b)]$$

It is convenient to see substitution as a form of β -reduction in λ -calculus, but without variable renaming:

$$(\lambda x. g(a, x)) @ h(b) \mapsto g(a, x) [x/h(b)]$$

Theorem (Knapik & Niwiński & Urzyczyn '02)

In the **safe fragment** of typed λ -calculus, β -reduction can be performed without variable renaming, that is, by substitution.

Corollary

In the safe typed λ -calculus, simultaneous β -reduction of redexes can be implemented by **MSO-interpretation** followed by **unfolding**.

It is convenient to see substitution as a form of β -reduction in λ -calculus, but without variable renaming:

$$(\lambda x. g(a, x)) @ h(b) \mapsto g(a, x) [x/h(b)]$$

Theorem (Knapik & Niwiński & Urzyczyn '02)

In the **safe fragment** of typed λ -calculus, β -reduction can be performed without variable renaming, that is, by substitution.

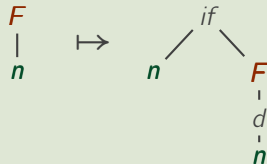
Corollary

In the safe typed λ -calculus, simultaneous β -reduction of redexes can be implemented by **MSO-interpretation** followed by **unfolding**.

 The above result applies also to infinitary terms!

A real application example for unfolding and λ -calculus!

Consider a safe **recursive program scheme** with functional variables

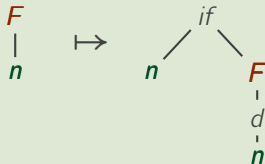


Abstraction of a program, e.g.

```
function Foo(n)
  if [n is prime] then
    return n
  else
    return Foo(divide (n))
```


A real application example for unfolding and λ -calculus!

Consider a safe **recursive program scheme** with functional variables and the **limit tree** generated from an initial axiom $F(9)$

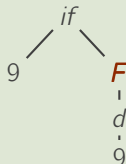
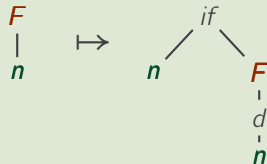


Abstraction of a program, e.g.

```
function Foo(n)
  if [n is prime] then
    return n
  else
    return Foo(divide (n))
```

A real application example for unfolding and λ -calculus!

Consider a safe **recursive program scheme** with functional variables and the **limit tree** generated from an initial axiom $F(9)$

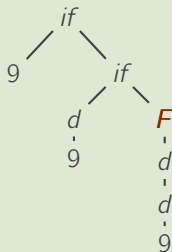
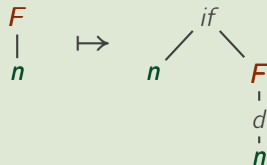


Abstraction of a program, e.g.

```
function Foo(n)
  if [n is prime] then
    return n
  else
    return Foo(divide (n))
```

A real application example for unfolding and λ -calculus!

Consider a safe **recursive program scheme** with functional variables and the **limit tree** generated from an initial axiom $F(9)$

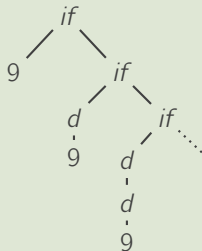
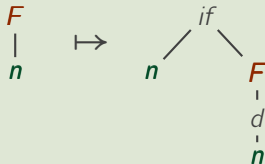


Abstraction of a program, e.g.

```
function Foo(n)
  if [n is prime] then
    return n
  else
    return Foo(divide (n))
```

A real application example for unfolding and λ -calculus!

Consider a safe **recursive program scheme** with functional variables and the **limit tree** generated from an initial axiom $F(9)$

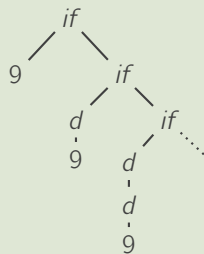
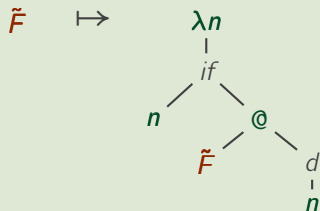
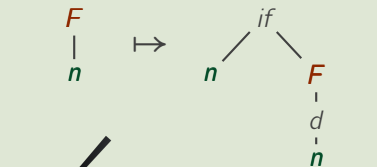


Abstraction of a program, e.g.

```
function Foo(n)
  if [n is prime] then
    return n
  else
    return Foo(divide (n))
```

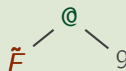
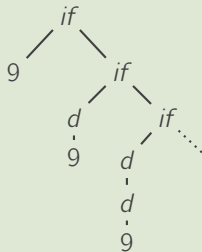
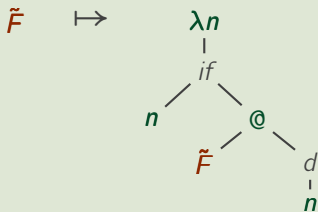
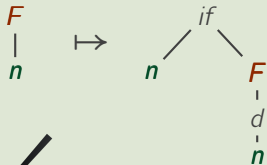
A real application example for unfolding and λ -calculus!

Consider a safe **recursive program scheme** with functional variables and the **limit tree** generated from an initial axiom $F(9)$



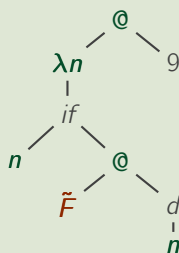
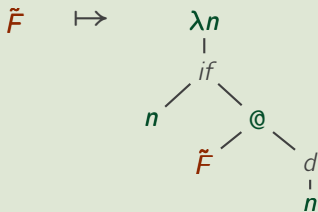
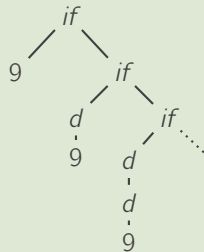
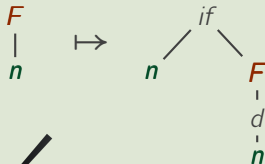
A real application example for unfolding and λ -calculus!

Consider a safe **recursive program scheme** with functional variables and the **limit tree** generated from an initial axiom $F(9)$



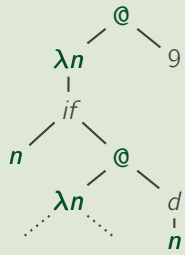
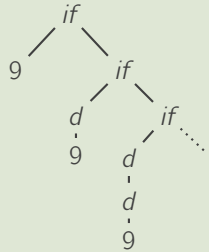
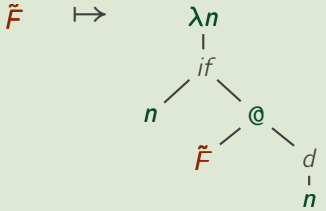
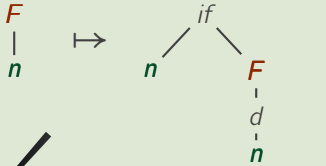
A real application example for unfolding and λ -calculus!

Consider a safe **recursive program scheme** with functional variables and the **limit tree** generated from an initial axiom $F(9)$



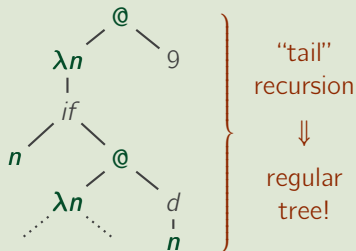
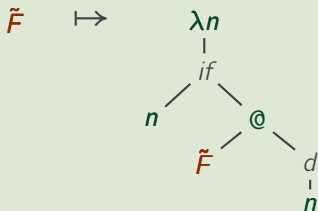
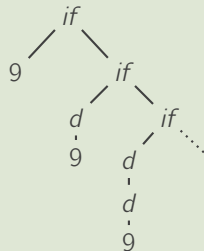
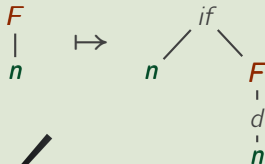
A real application example for unfolding and λ -calculus!

Consider a safe **recursive program scheme** with functional variables and the **limit tree** generated from an initial axiom $F(9)$



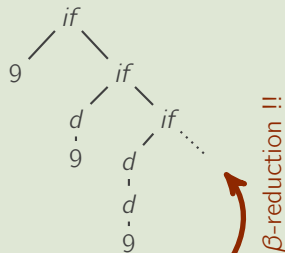
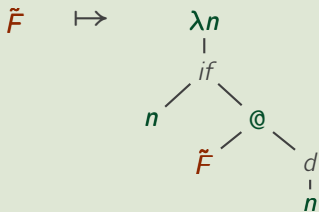
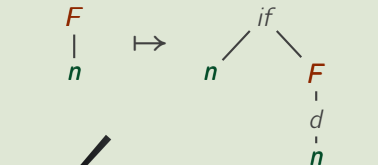
A real application example for unfolding and λ -calculus!

Consider a safe **recursive program scheme** with functional variables and the **limit tree** generated from an initial axiom $F(9)$

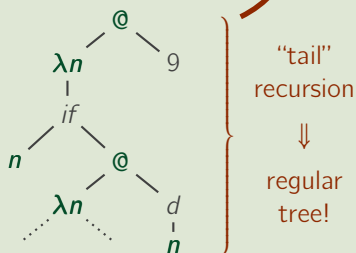


A real application example for unfolding and λ -calculus!

Consider a safe **recursive program scheme** with functional variables and the **limit tree** generated from an initial axiom $F(9)$



β -reduction !!



Previous ideas apply to any scheme with higher-order variables:

Theorem (Knapik & Niwiński & Urzyczyn '02)

Limit trees of **safe order- n recursive program schemes** are in the **level n of Caucal hierarchy**, hence they have decidable MSO theories.

Previous ideas apply to any scheme with higher-order variables:

Theorem (Knapik & Niwiński & Urzyczyn '02)

Limit trees of **safe order- n recursive program schemes** are in the **level n of Caucal hierarchy**, hence they have decidable MSO theories.

Theorem (Parys '12)

Starting from level 2, safety is a genuine restriction for λ -calculus.

Previous ideas apply to any scheme with higher-order variables:

Theorem (Knapik & Niwiński & Urzyczyn '02)

Limit trees of **safe order- n recursive program schemes** are in the **level n of Caucal hierarchy**, hence they have decidable MSO theories.

Theorem (Parys '12)

Starting from level 2, safety is a genuine restriction for λ -calculus.

Theorem (Ong '06)

Limit trees of **unsafe order- n recursive program schemes** have also decidable MSO theories.

▶ Next