Transfer Theorems

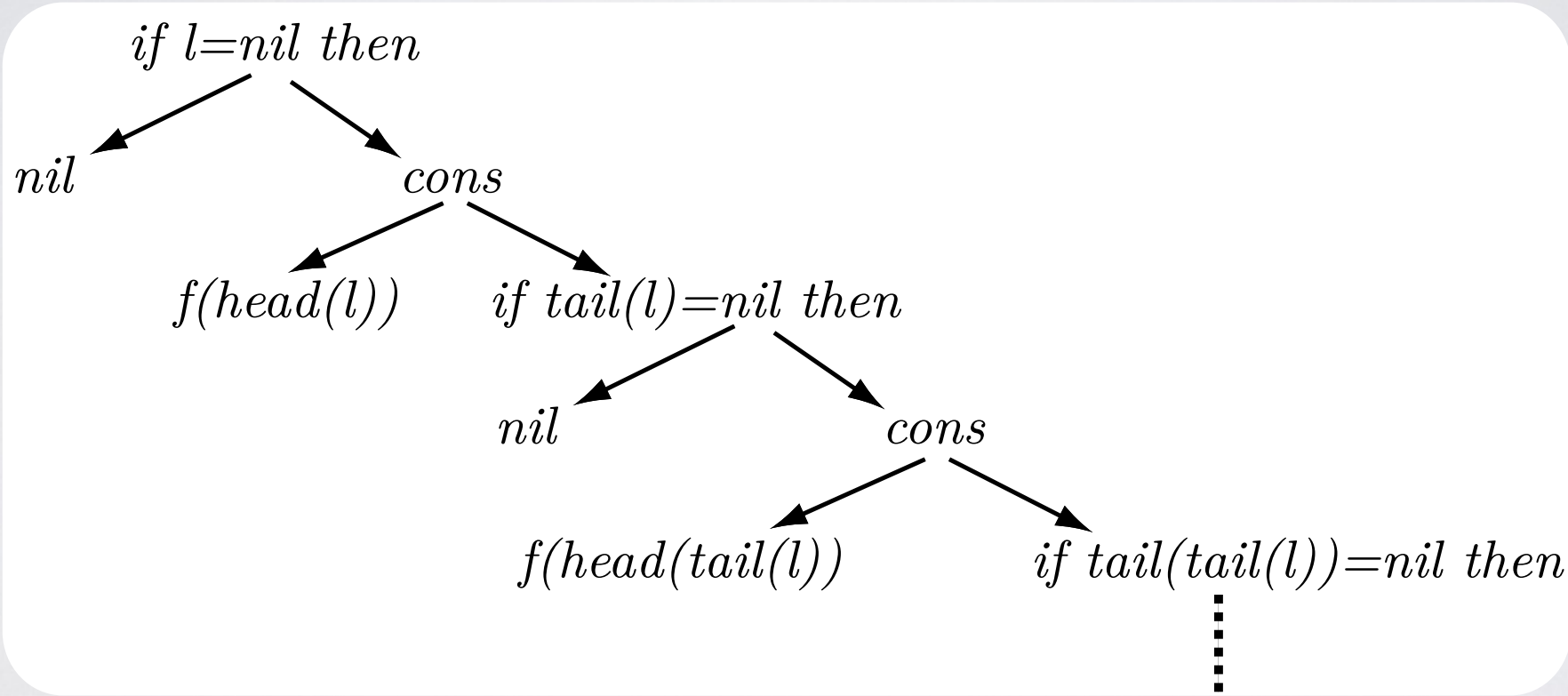
Igor Walukiewicz

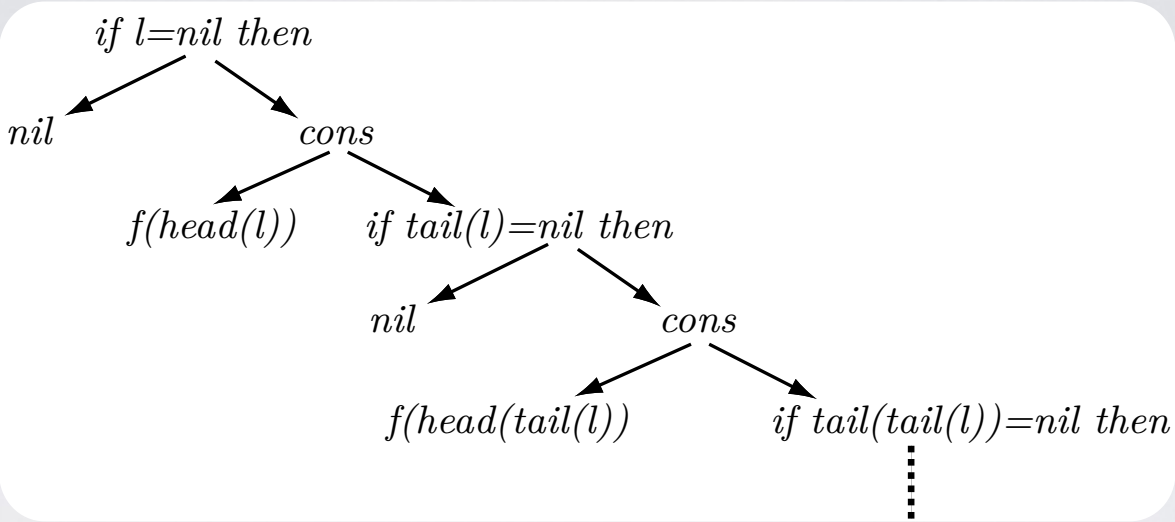
Bordeaux University

$$map(f, l) \equiv \textbf{if } l = nil \textbf{ then } nil \textbf{ else } \textbf{cons}(f(\textbf{head}(l)), map(f, \textbf{tail}(l)))$$

$$map(f, (a, b, c)) = (f(a), f(b), f(c))$$

$$map(f, l) \equiv \textbf{if } l = nil \textbf{ then } nil$$
$$\textbf{else cons}(f(\textbf{head}(l)), map(f, \textbf{tail}(l)))$$





Such trees are interesting because

- They reflect a part of the semantics of a program.
- They have decidable MSOL theory.
- Interesting properties can be expressed in MSOL:
  - All elements in the result are in the range of f

While-programs

$$x := e \mid \texttt{if } x = 0 \texttt{ then } I_1 \texttt{ else } I_2 \mid \texttt{while } x > 0 \texttt{ do } I$$

variables range over $\mathbb{N}$ and e are arithmetic expressions

- While-programs are Turing powerful.
- Does this mean that all other programming concepts are obsolete?
- Schemes give a way to show that they are not:
  - There is a recursive scheme whose tree cannot be generated by a scheme of a while program.

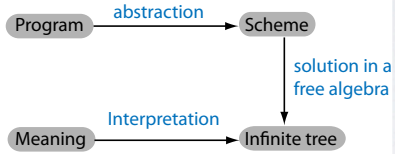
Some history on the logical side

- Z. Manna “Mathematical Theory of Computation”, 1974
- Algorithmic Logic [A. Salwicki 68]
- Dynamic Logic [V. Pratt 76]
- Propositional Dynamic Logic [Parikh 78, Fisher & Ladner 79]
- Propositinal Dynamic Logic with Deadlock [Harel 84]
- Propositional Mu-calculus [Kozen 83]

- Pnueli [76]
- Mana & Pnueli [80]

Schemes

+ Ianov'58 "The logical schemas of algorithms"
+ Park PhD'68 Recursive schemes
+ Scott, Elgot



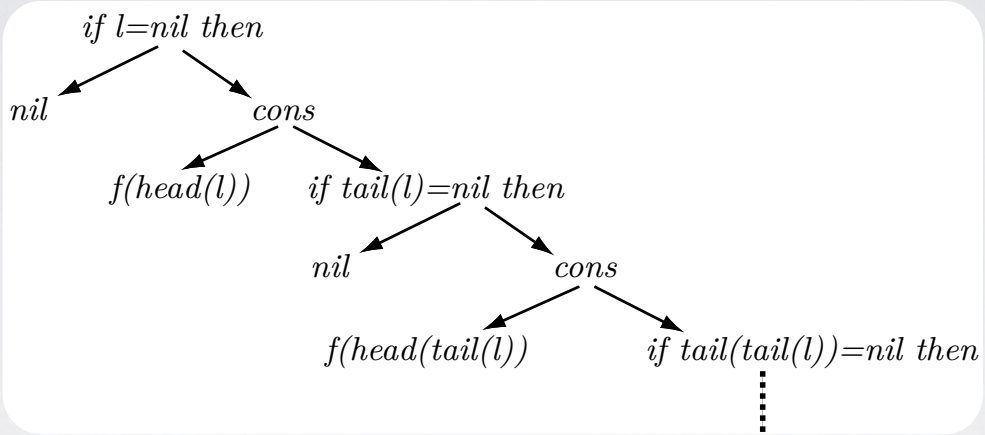
+ Milner'73 Plotkin'77 PCF

Languages, Higher-order pushdowns

+ Aho'68 indexed languages
+ Maslov'74 '76 higher-order indexed languages and higher order pushdown automata.

+ Courcelle'76 for trees: 1-st order schemes=CFL
+ Engelfriet Schmidt'77 IO/OI
+ Damm'82 for languages: rec schemes= higher-order pusdowns
+ Kanpik Niwinski Urzyczyn'02 Safe schemes = higher-order pusdown
+ Senizergues'97 Equivalence of 1st order schemes is decidable
  +Statman'04 Equivalence of PCF terms is undecidable
  +Loader'01: Lambda-definability is undecidable
+ Ong'06: Decidability of MSOL theory

Two main algorithmic problems



Deciding equality of schemes:

Do two schemes generate the same trees?

Deciding MSOL theory for schemes:

Does a given MSOL formula hold in a tree generated by a scheme?

Ad equality: Decidable for schemes of order 1 [Senizergues]

Ad MSOL: Decidable [Ong]

In this talk:

What are trees generated by higher-order schemes?

We will consider:

- Operations on (colored) graphs $\mathcal{F} : \mathit{Graphs} \to \mathit{Graphs}$
- Monadic Second-Order Logic

$$P(x) \mid E(x,y) \mid \varphi \vee \psi \mid \neg\varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

In this talk

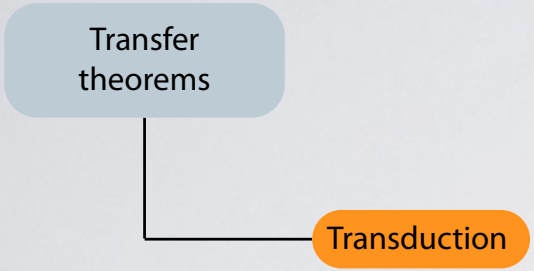
Consider an operation $\mathcal{F}$ on graphs

Transfer property for $\mathcal{F}$
For every φ one can effectively construct $\widehat{\varphi}$, s.t., for every G:

$$\mathcal{F}(G) \models \varphi \qquad \text{iff} \qquad G \models \widehat{\varphi}.$$

We say in this case that $\mathcal{F}$ is **MSO-compatible**.

We will show that one can understand trees generated by schemes in terms of such an operation.


Transfer
theorems
Transduction

MSO interpretations

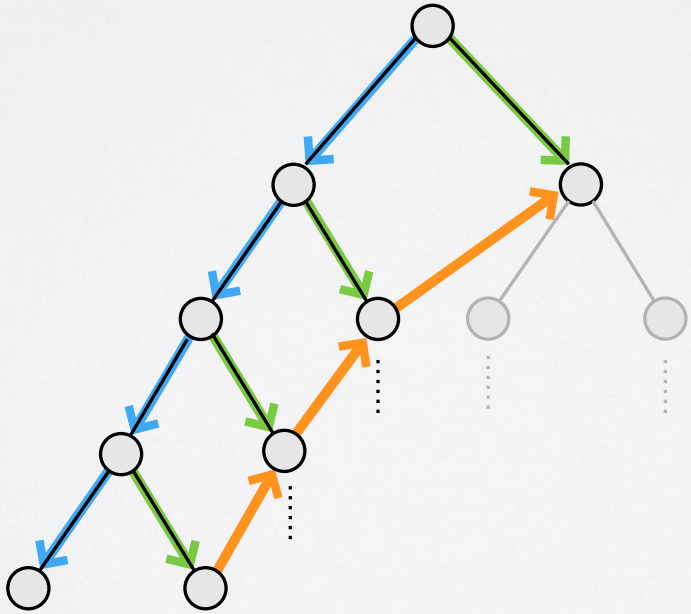
Graph with labeled edges: $G = \langle V, \{E_a\}_{a \in \Sigma} \rangle$

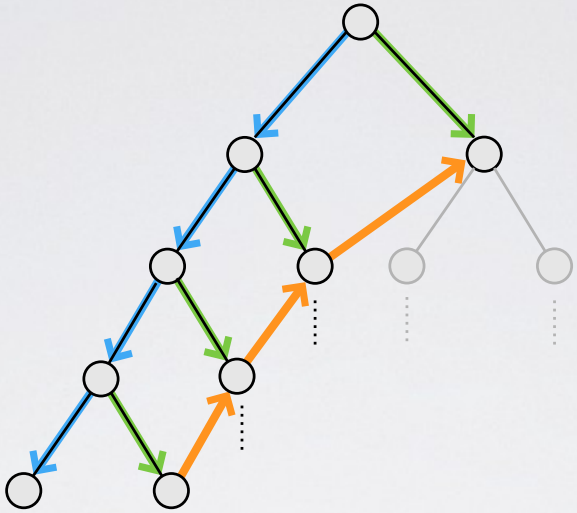
Graph with edge labels from Σ

$\downarrow$

graph with edge labels form Δ

determined by formulas: $\{\varphi_a(x, y)\}_{a \in \Delta}$





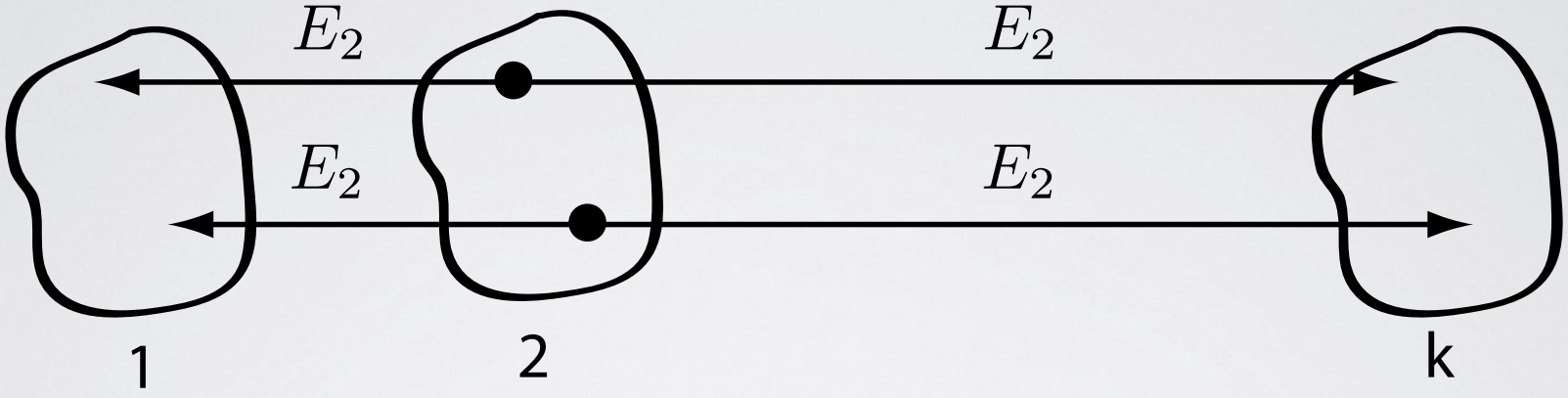
MSO-interpretations are MSO compatible.
For every φ one can effectively construct $\widehat{\varphi}$, s.t., for every M:

$$\mathcal{I}(M) \vDash \varphi \qquad \text{iff} \qquad M \vDash \widehat{\varphi}.$$

$$\widehat{\varphi} \equiv \varphi[\varphi_a(x,y) \mapsto E_a(x,y)]$$

k-COPYING

Duplicating k-times a graph $G = \langle V, \{E_a\}_{a \in \Sigma} \rangle$.



$G' = \langle V', \{E'_a\}_{a \in \Sigma}, \{E_i\}_{i \in [k]} \rangle$; where

- $V' = V \times [k]$;
- $E'_a((v,i),(w,i))$ for $(v,w) \in E_a$ and $i \in [k]$;
- $E_i((v,i),(v,j))$ for $v, w \in V$ and $j \in [k]$.

The operation of k-copying is MSO compatible.

MSO-transductions

MSO-transduction is a sequence of copying and MSO interpretations

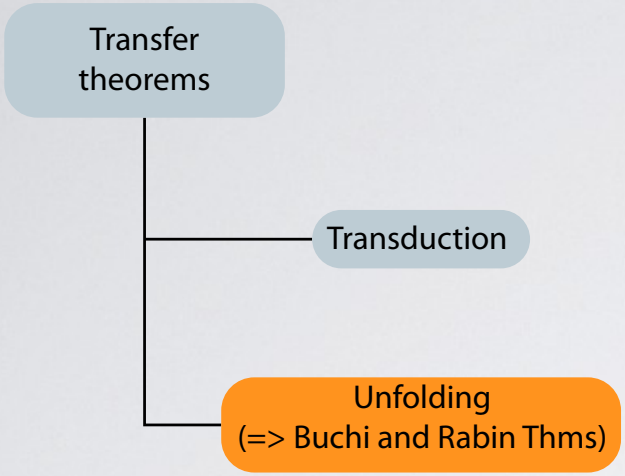
Fact: MSO-transduction is MSO compatible.

$$\begin{array}{ccccccccccc} M_0 & \xrightarrow{copy} & M_1 & \xrightarrow{\mathcal{I}} & M_2 & \dots & \xrightarrow{copy} & M_{k-1} & \xrightarrow{\mathcal{I}} & M_k \\ \varphi_0 & \longleftarrow & \varphi_1 & \longleftarrow & \varphi_2 & \dots & \longleftarrow & \varphi_{k-1} & \longleftarrow & \varphi_k \end{array}$$

$$M_0 \models \varphi_0 \quad \text{iff} \quad M_k \models \varphi_k$$

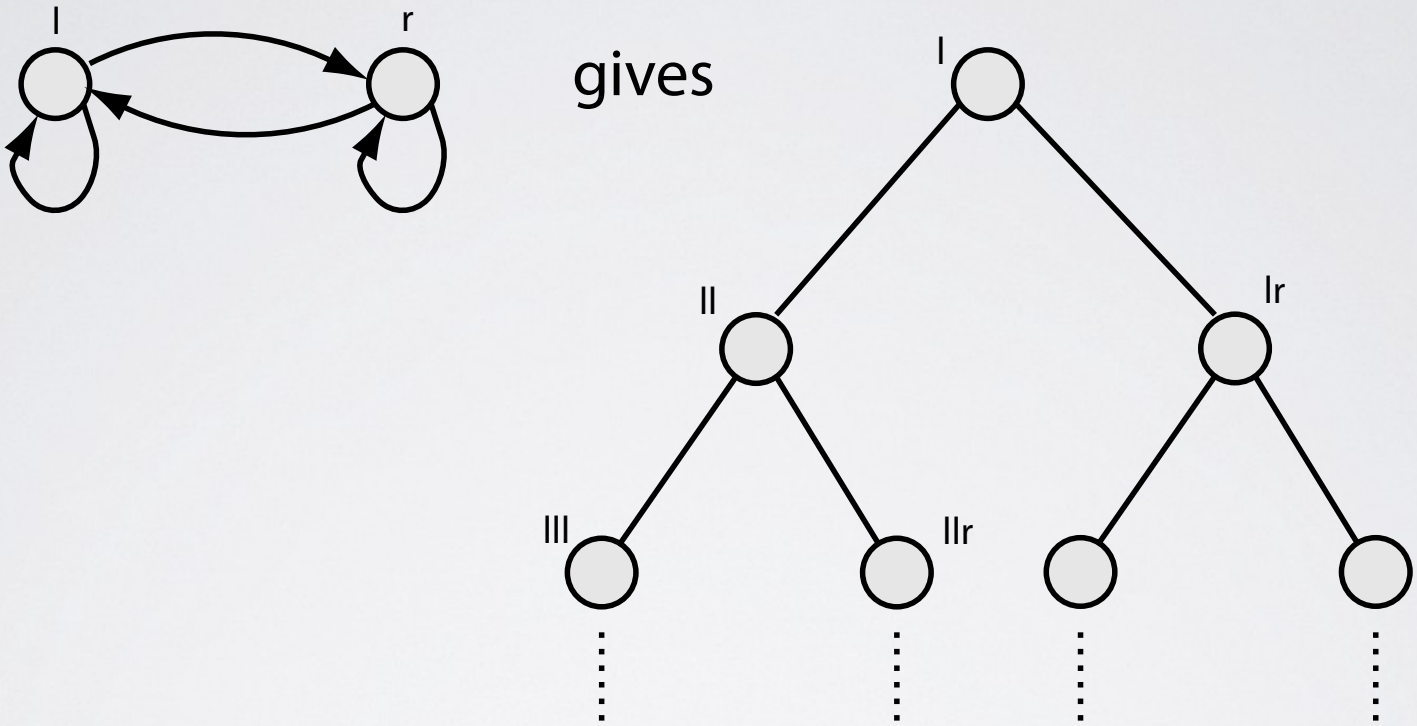
Example: from one node graph we can construct any finite graph.

Remark: Actually it suffices to do one copying and one interpretation.

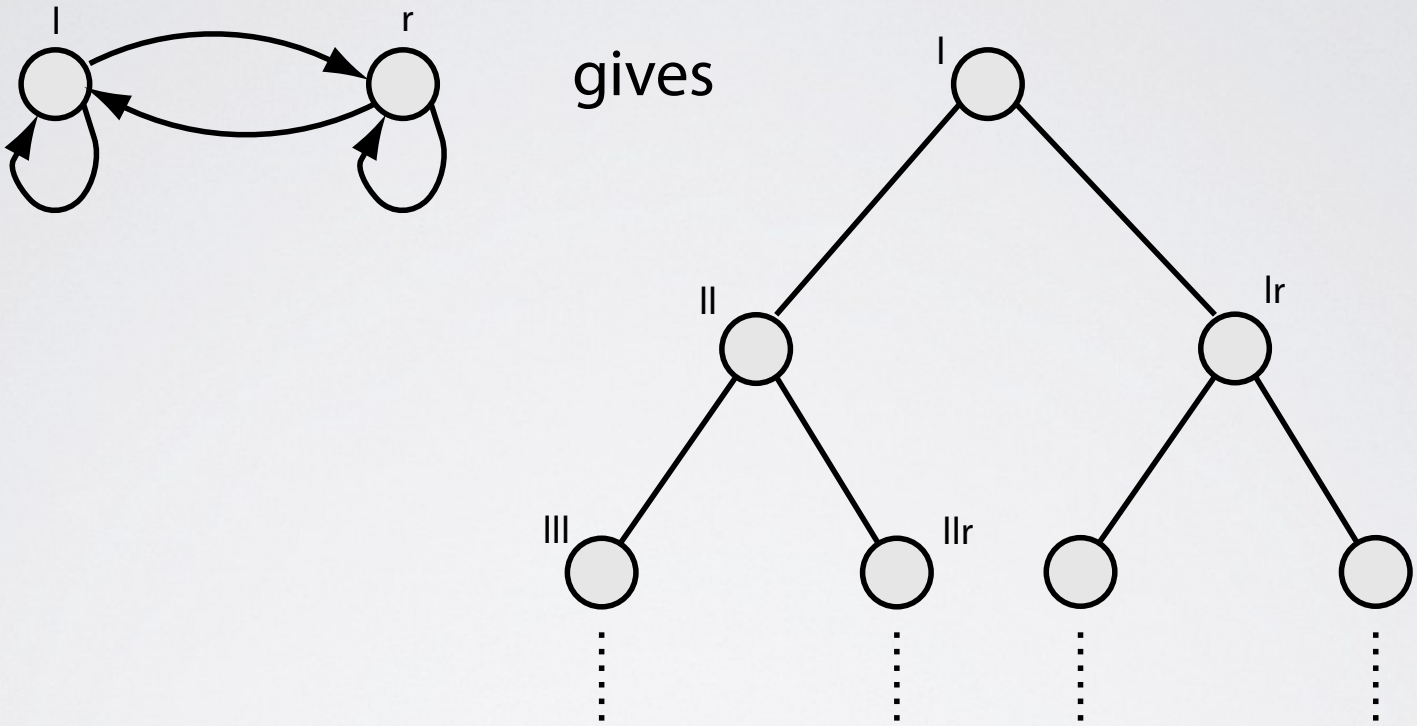

Transfer
theorems
Transduction
Unfolding
(=> Buchi and Rabin Thms)

Unfolding: the tree of all the paths in the graph from a given node.

Unfolding: the tree of all the paths in the graph from a given node.



Unfolding: the tree of all the paths in the graph from a given node.



$Unf(G, v_0) = \langle V^U, \{E_a^*\}_{a \in \Sigma} \rangle$ where

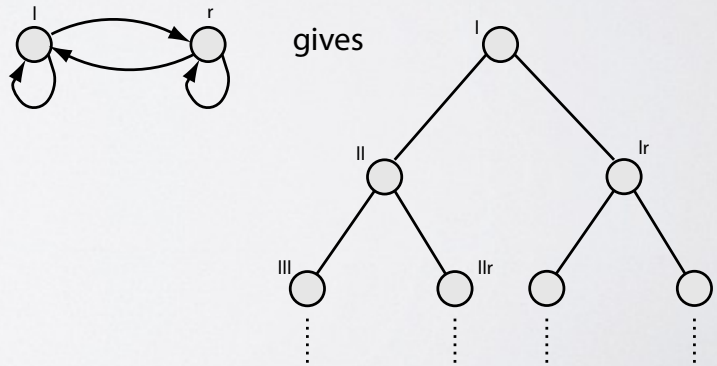
- $V^U =$ paths in G starting from v_0
- $E_a^*(\mathbf{w}v, \mathbf{w}vu)$ if $E_a(v, u)$, and $\mathbf{w} \in V^U$.

Theorem[Courcelle & W.,Muchnik]:
Unfolding is MSO-compatible.
For every $\varphi(x)$ there is (effectively) $\widehat{\varphi}(v_0)$ such that
for every graph G and its vertex v_0:

$$G \models \widehat{\varphi}(v_0) \qquad \text{iff} \qquad Unf(G) \models \varphi(v_0)$$

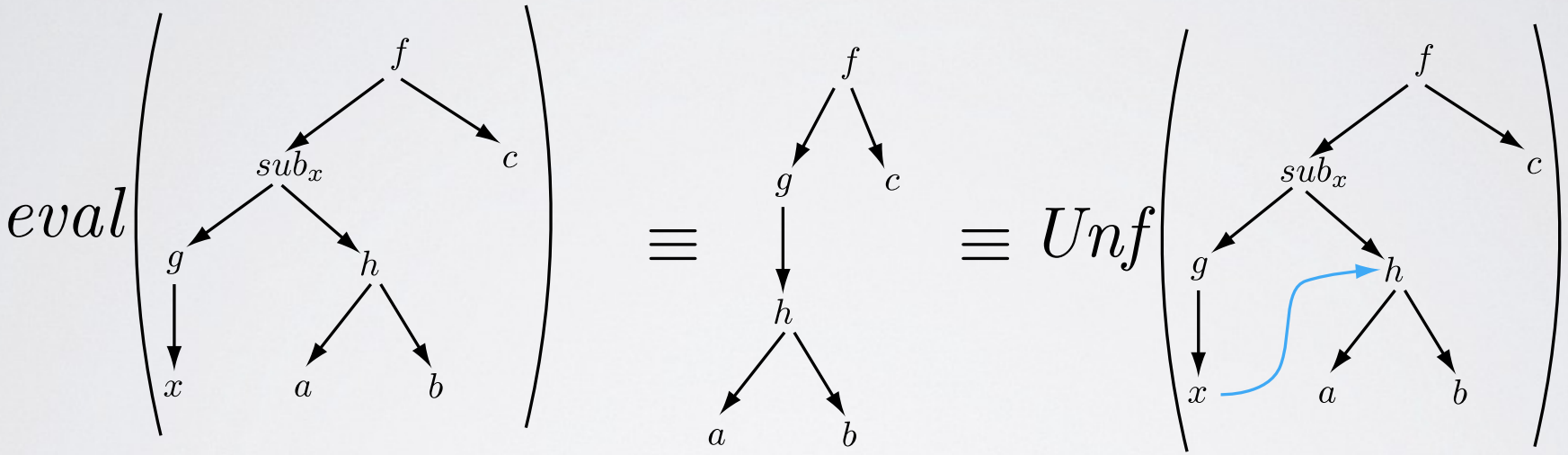
Remark 1: Unfolding cannot be defined by a transduction.

Remark 2: MSO-compatibility of the unfolding implies Büchi and Rabin's Theorems.

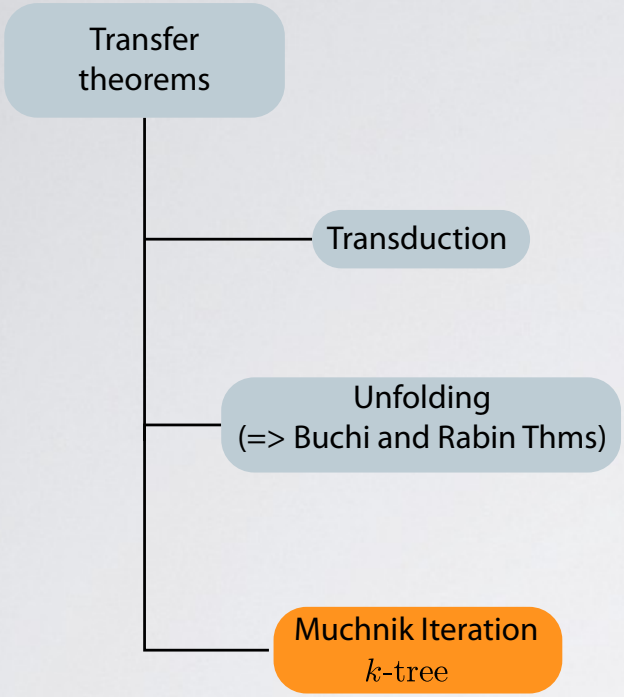


Tree with substitutions: function symbols $a, f, g, \ldots$; variables $x, y, \ldots$; and explicit substitutions sub_x.

$$eval(sub_x(s,t)) = s[t/x]$$

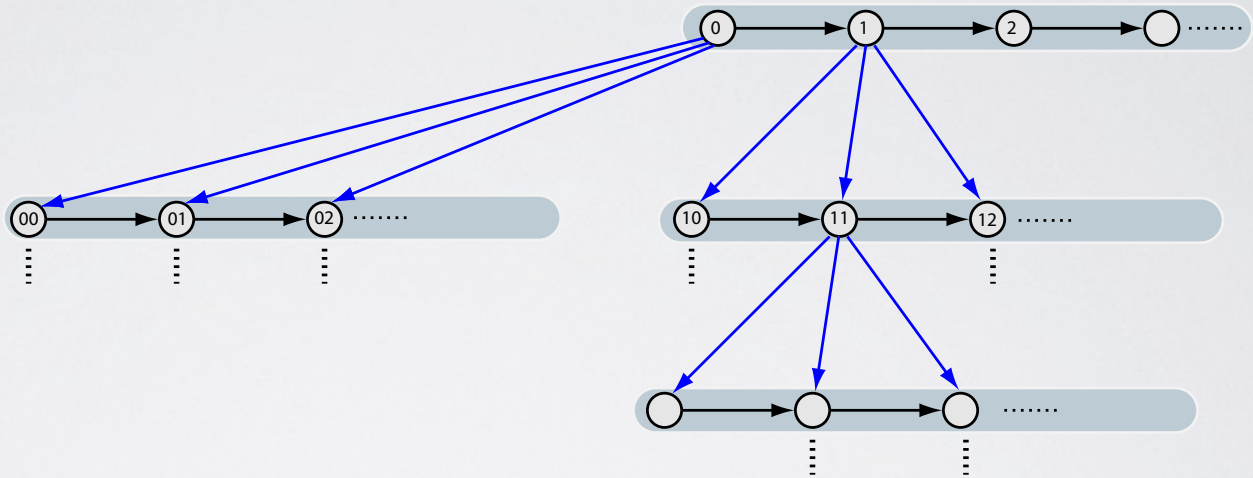


Theorem[Courcelle & Knapik]: For fixed finite set of variables: $eval$ is MSO-compatible


Transfer theorems
Transduction
Unfolding (=> Buchi and Rabin Thms)
Muchnik Iteration
k-tree

Stupp iteration

$St($  $)$ is



$St(G) = \langle V^+, \{E_a^*\}_{a\in\Sigma}, son\rangle$
where for $\mathbf{w} \in V^*$, $u, v \in V$:

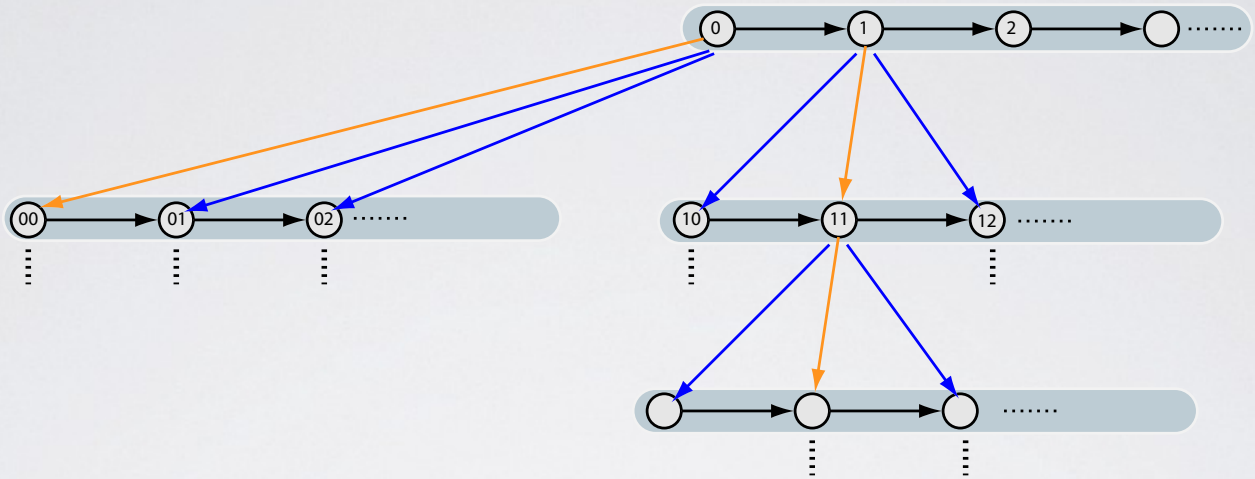
- $son(\mathbf{w}, \mathbf{w}v)$,
- $E_a^*(\mathbf{w}u, \mathbf{w}v)$ when $E_a(u, v)$.

Remark 1: Stupp iteration of the two node graph gives two full binary infinite trees.

Remark 2: Unfolding of a graph may not be definable in the Stupp iteration of the graph.

Remark 3: Stupp iteration of the full binary tree is MSO definable in the full binary tree.

Muchnik iteration

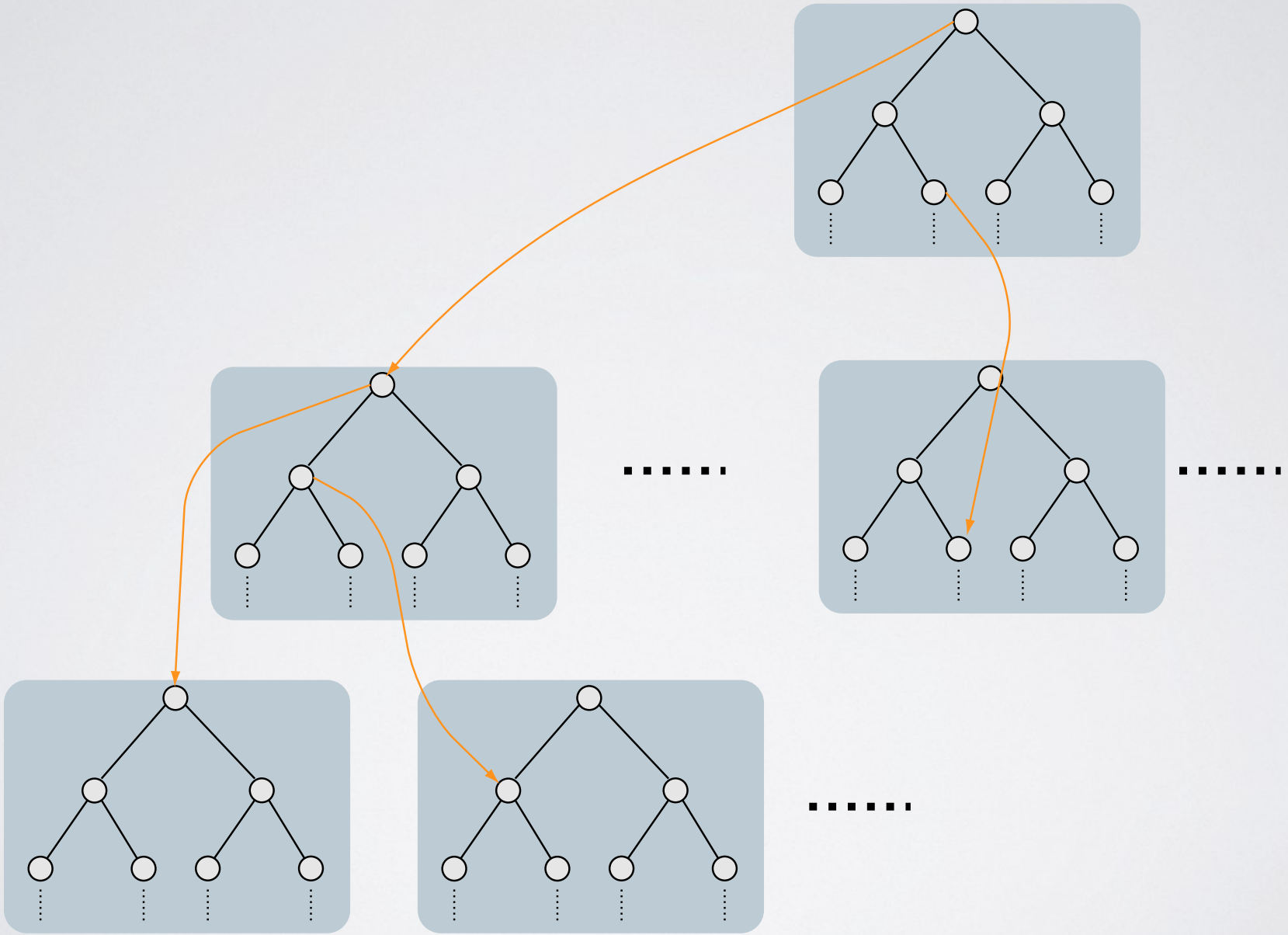


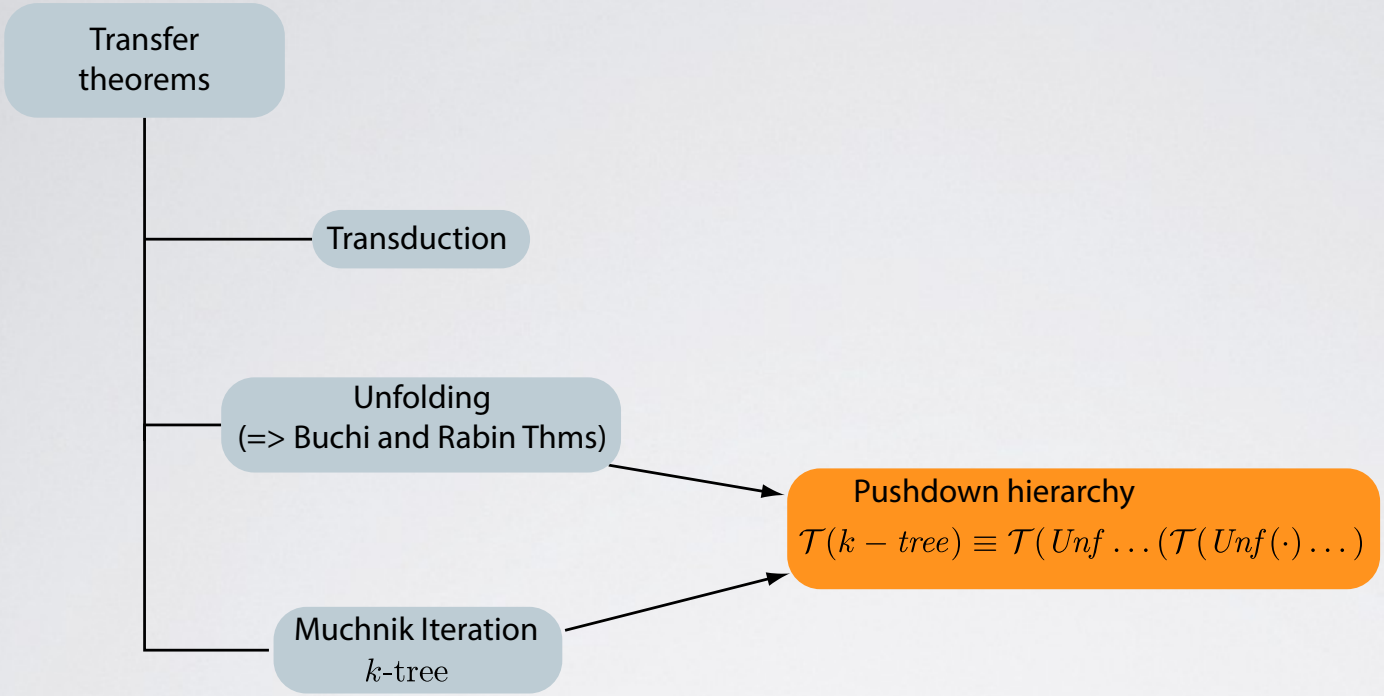
$G^+ = \langle V^+, \{E^*\}_{a\in\Sigma}, E_\#, son\rangle$

- $E_\#(wu, wuu)$ for $w \in V^*$ and $u \in V$.

Theorem[Muchnik,W.]: Muchnik iteration is MSO-compatible.

2-tree: Muchnik iteration of the full binary tree.




Transfer theorems
Transduction
Unfolding (=> Buchi and Rabin Thms)
Muchnik Iteration
k-tree
Pushdown hierarchy
$\mathcal{T}(k-tree) \equiv \mathcal{T}(Unf \ldots (\mathcal{T}(Unf(\cdot)\ldots)$

Pushdown hierarchy

- Level-0: finite graphs
- Level-k: MSO-transductions of k-tree.

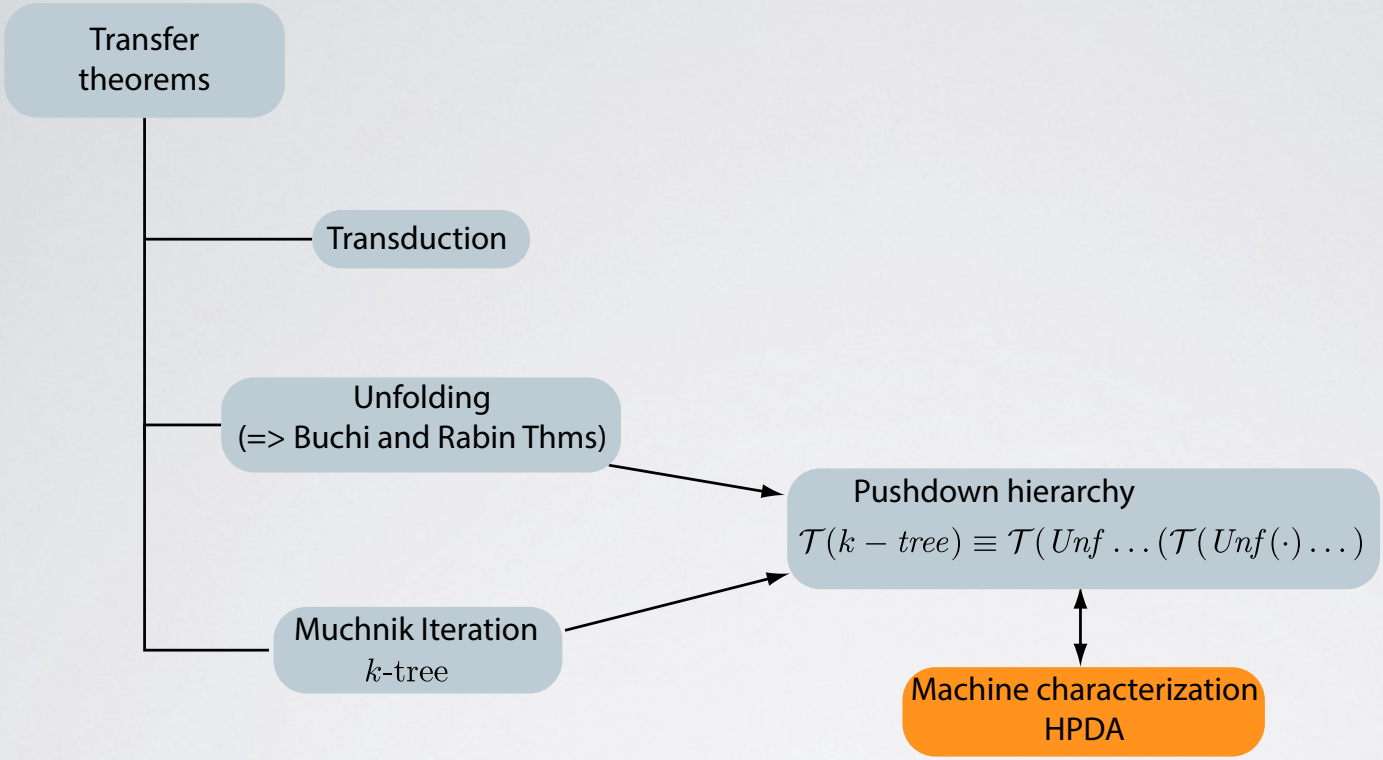
Equivalently:

- Level-k: MSO transductions of unfoldings of Level-$(k-1)$ graphs.

$$\mathcal{T}(k-tree) \equiv \overbrace{\mathcal{T}(Unf(\dots(\mathcal{T}(Unf}^{k}(finite\ graph)\dots)$$

Cor: All graphs in the pushdown hierarchy have decidable MSO-theory.

Thm [Engelfriet, Carayol & Wöhrle]**:** Pushdown hierarchy is strict.


Transfer theorems
Transduction
Unfolding (=> Buchi and Rabin Thms)
Muchnik Iteration
k-tree
Pushdown hierarchy
$\mathcal{T}(k-tree) \equiv \mathcal{T}(Unf \ldots (\mathcal{T}(Unf(\cdot)\ldots)$
Machine characterization HPDA

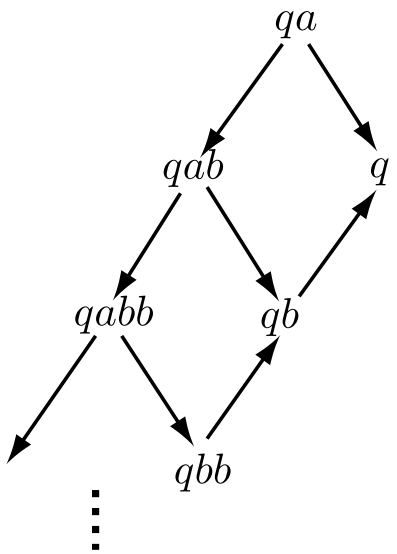
General idea

A **graph of configurations** of a machine:

- nodes are configurations of the machine;
- edges represent a step of the computation.

Finite automaton: its graph of configurations is just graph of the automaton

Pushdown automaton:
nodes $qab\ldots b$
edges
$qa\mathbf{w} \to q\mathbf{w}$
$qa\mathbf{w} \to qab\mathbf{w}$
$qb\mathbf{w} \to q.$



2-ND ORDER STACK: EXAMPLE

A 2-**stack** is a stack of stacks. $[a_1^1 \dots a_{k_1}^1][a_1^2 \dots a_{k_2}^2] \dots [a_1^n \dots a_{k_n}^n]$

New operation of copying the top-most stack:
$q[w_1] \dots [w_i] \to q[w_1][w_1] \dots [w_i]$.

$$
\begin{array}{llllllll}
q_1[a] & \to & q_1[aa] & \to & \cdots & \to & q_1[a^k] \to & \\
q_2[a^k] & \to & q_2[a^{k-1}][a^k] & \to & \cdots & \to & q_2[a][aa] \dots [a^k] \to & \\
q_3[a][aa] \dots [a^k] & \to & q_3[aa] \dots [a^k] & \to & \cdots & \to & q_3[a^k] \to q_3\bot &
\end{array}
$$

A maximal paths are of the form $q_1^k q_2^k q_3^k$.

Remark: The 2-stack gives additional power.

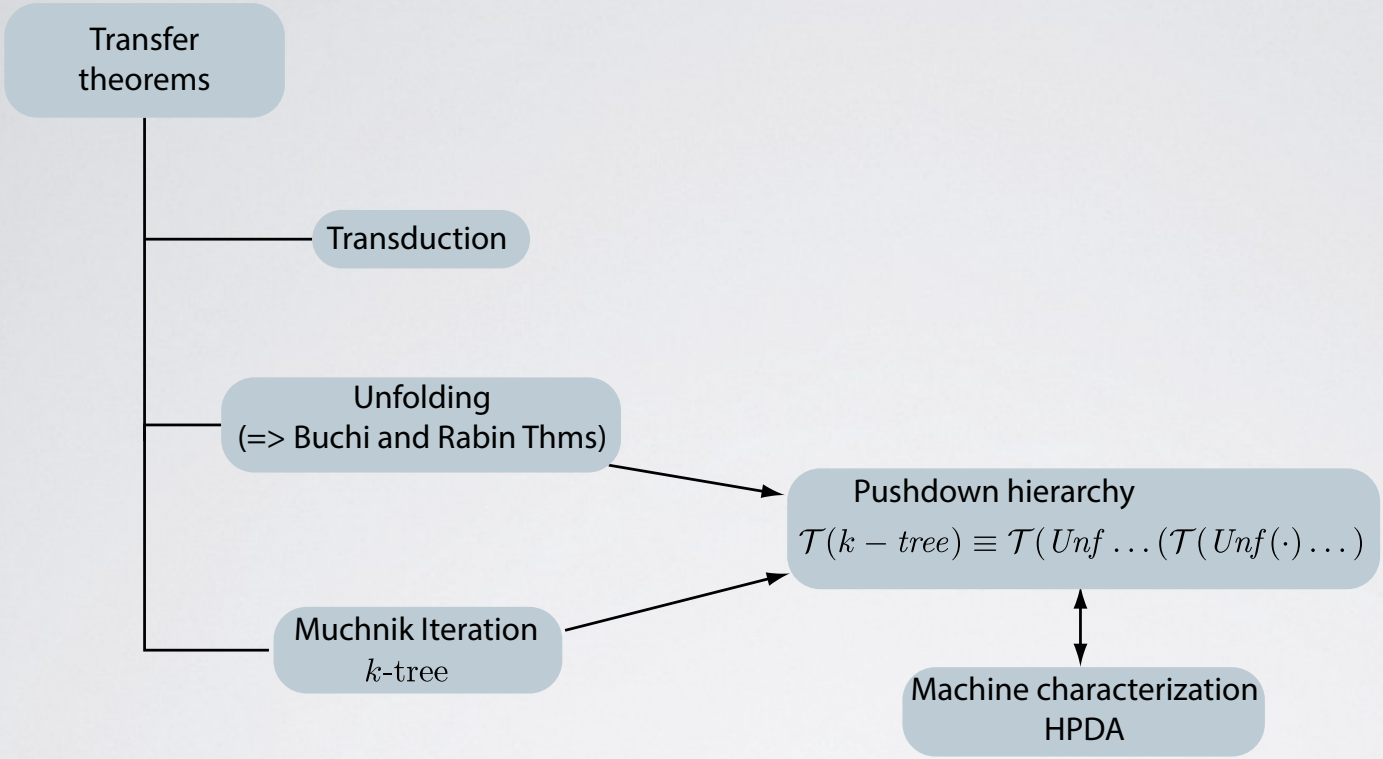
Remark: The above automaton recognizes $\{a^k b^k c^k : k \in \mathbb{N}\}$.

Higher order pushdowns ≡ Pushdown hierarchy

- Configuration graph of a pushdown automaton is interpretable in a tree
- Configuration graph of a k-pushdown automaton is interpretable in a k-tree.

Cor: All these graphs have decidable MSO-theory.

Thm[Carayol & Wöhrle]:
Graphs of pushdown hierarchy level k are configuration graphs of k-th order pushdown automata. (when ε-transitions are contracted).



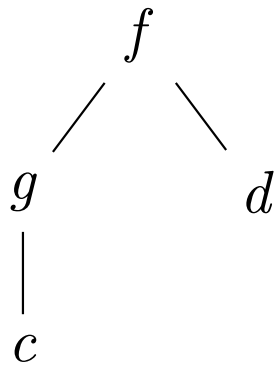
$\lambda Y - calculus$

Simply typed λ-calculus with fixpoints

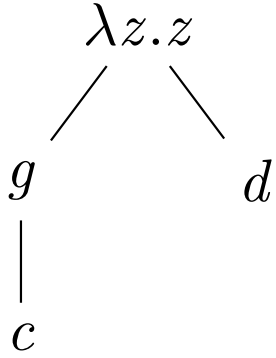
- Types: 0 is a type, and $\alpha \to \beta$ is a type if α, β types.
- Constants: c^α of type α.
- Terms: $c^\alpha, \quad x^\alpha, \quad MN, \quad \lambda x^\alpha.M$.

Example: $c, d : 0, \quad g : 0 \to 0, \quad f : 0 \to 0 \to 0$

$f(gc)d : 0$



$\lambda z.z(gc)d : (0 \to 0 \to 0) \to 0$



β-reduction: $(\lambda x.M)N \to_\beta M[N/x]$

$(\lambda x.f(gx)d)c \to_\beta f(gc)d$

$(\lambda z.z(gc)d)(\lambda xy.y) \to_\beta (\lambda xy.y)(gc)d \to_\beta d$

Substitution is as in logic: one should avoid variable capture

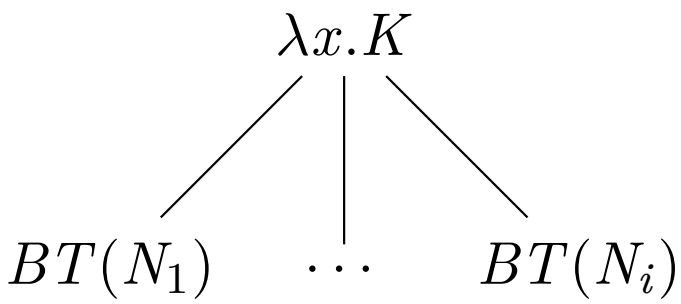
$(\lambda h.\lambda x.g(hx))(fx) \to_\beta \lambda y.g(fxy)$

and not $\lambda x.g(fxx)$

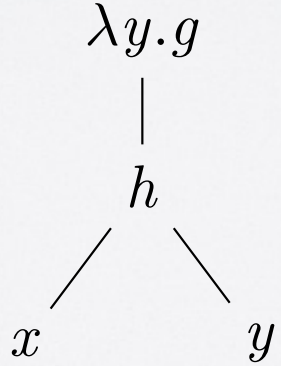
$f : 0 \to 0 \to 0, \quad g, h : 0 \to 0$

A **Böhm tree of a term M:**

- We reduce M to head normal form:
  $M \to_\beta^* \lambda\vec{x}.KN_1 \ldots N_i$ with K a variable or a constant.
- $BT(M)$ is



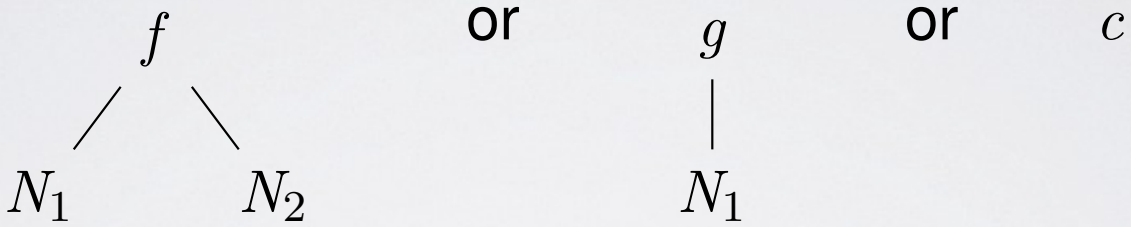
Böhm tree of $(\lambda y.g(hxy))$ is



Where are trees?

$$c : 0,\ g : 0 \to 0,\ f : 0 \to 0 \to 0$$

If $M : 0$ is a closed term, and M in head normal form then $M \equiv KN_1 \dots N_i$ with K a constant. So it is either:



with $N_0, N_1 : 0$. Hence $BT(M)$ is a ranked tree.

Order of type: $\mathit{Ord}(0) = 0$, $\mathit{Ord}(\alpha \to \beta) = \max(\mathit{Ord}(\alpha) + 1, \mathit{Ord}(\beta))$.

First order signature: all constants of order ≤ 1.

Remark: For closed $M : 0$ over a first order signature $BT(M)$ is a ranked tree.

λY-calculus

We add constants $Y^{(\alpha\to\alpha)\to\alpha}$ and ω^{α}, for every type α.

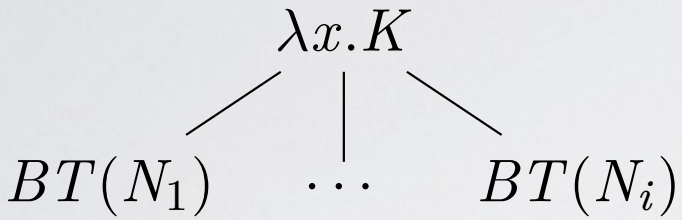
New reduction rule $YM \to_\delta M(YM)$.

Example: YM with $M = (\lambda x.ax)$

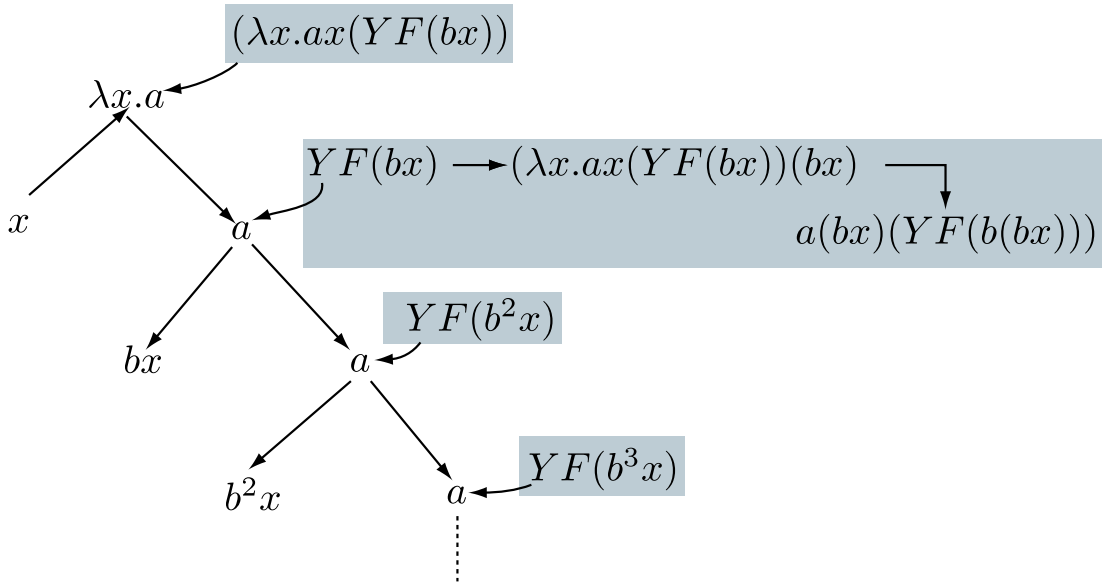
$$\begin{aligned} YM &\to_\delta M(YM) \equiv (\lambda x.ax)(YM) \\ &\to_\beta a(YM) \\ &\to_\delta a(M(YM)) \\ &\to_\beta a(a(YM)) \to \dots \end{aligned}$$

A **Böhm tree of a λY-term** M is:

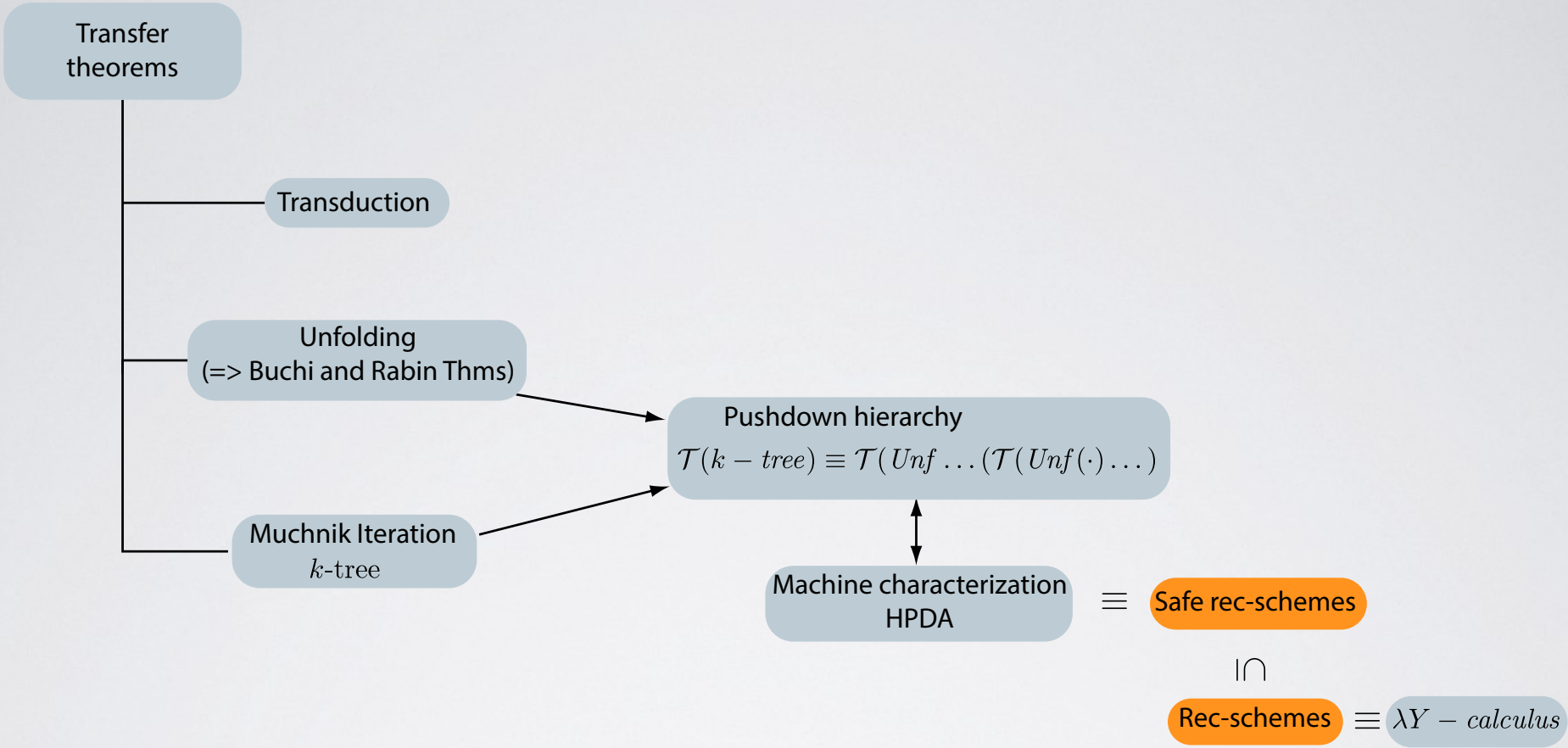
- If M has no head normal form then ω^α.
- Otherwise $\lambda\vec{x}.KN_1 \dots N_i$ is the head normal form and $BT(M)$ is



$Y(\lambda F.\lambda x.ax(F(bx))) \;:\; 0 \to 0$



For closed terms of type 0 over first-order signatures, Böhm tree is a tree.


Transfer theorems
Transduction
Unfolding (=> Buchi and Rabin Thms)
Muchnik Iteration
k-tree
Pushdown hierarchy
$\mathcal{T}(k-tree) \equiv \mathcal{T}(Unf \ldots (\mathcal{T}(Unf(\cdot)\ldots)$
Machine characterization HPDA
$\equiv$
Safe rec-schemes
$\cap$
Rec-schemes
$\equiv \lambda Y - calculus$

Recursive program schemes

[Indermark, MFCS'76]

First example

- $F \equiv \lambda x.\ \mathbf{if}\ x = 0\ \mathbf{then}\ 1\ \mathbf{else}\ F(x-1) \cdot x.$
- Abstract form: $F = \lambda x.\ c\ (zx)\ a\ (m\ (F(px))\ x).$

Another program with the same abstract form:
$Rev \equiv \lambda x.\ \mathbf{if}\ x = \mathrm{nil}\ \mathbf{then}\ \mathrm{nil}\ \mathbf{else}\ Rev(\mathrm{tl}(x)) \cdot hd(x)$

Second-order scheme

$map \equiv \lambda f.\lambda l.\ \mathbf{if}\ l = nil\ \mathbf{then}\ nil\ \mathbf{else}\ \mathbf{cons}(f(\mathbf{head}(l)), map(f, \mathbf{tail}(l)))$

Order of a scheme: maximal order of a "nonterminal".

Semantics

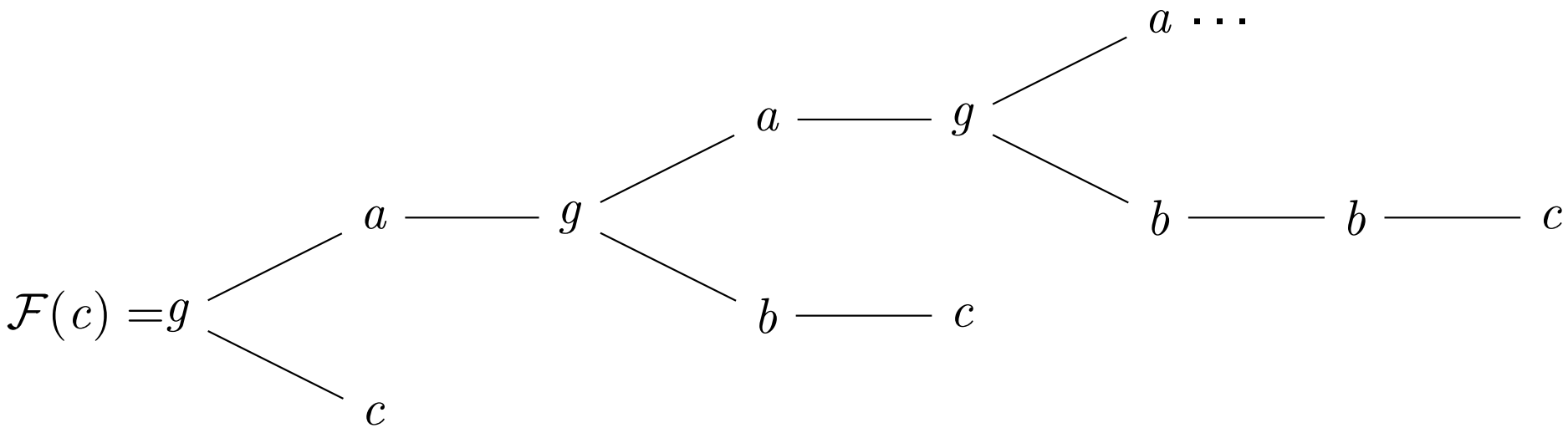
Example:

$$\mathcal{F} = \lambda x.\ a(\mathcal{F}(bx))$$

$$\mathcal{F}(c) \to a(\mathcal{F}(bc)) \to a(a(\mathcal{F}(b(bc)))) \to \dots$$

Semantics as a tree of execution

$$\mathcal{F} = \lambda x.\ g\ (a(\mathcal{F}(bx)))\ x$$



Recursion schemes $\equiv \lambda Y$-calculus

$$\begin{aligned} F_1 &= \lambda \vec{x}.M_1 \\ &\vdots \\ F_n &= \lambda \vec{x}.M_n \end{aligned}$$

$$\begin{aligned} T_1 &= Y(\lambda F_1.M_1) \\ T_2 &= Y(\lambda F_2.M_2)[T_1/F_1]) \\ &\vdots \\ T_n &= Y(\lambda F_n.(\ldots((M_n[T_1/F_1])[T_2/F_2])\ldots)[T_{n-1}/F_{n-1}]) \end{aligned}$$

Fact

The tree generated from F_n is $BT(T_n)$.
There is also a translation from λY-terms to schemes.

Theorem[Courcelle]:
The meanings of 1-st order recursive schemes $\equiv$ unfoldings of pushdown graphs.

Theorem[Knapik, Niwiński & Urzyczyn]:
n-th order safe schemes $\equiv$ unfoldings of n-th order pushdown graphs.

Safe $\approx$ no parameters in recursion $\approx$ no problems with static links

Safety

Variables that occur free in a safe λ-term have orders not smaller than that of the term itself.

Safe $\Rightarrow$ no need to perform variable renaming when doing β-reduction.

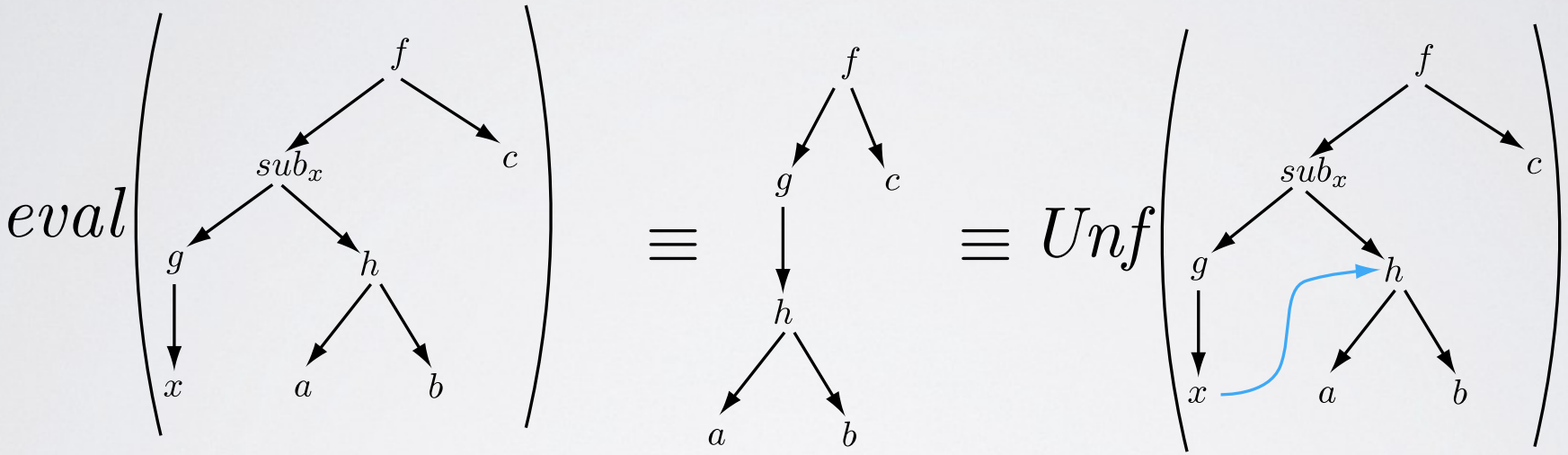
$$(\lambda x^{\alpha}.M)N^{\alpha} \rightarrow_{\beta} M[N^{\alpha}/x^{\alpha}]$$

$(\lambda y^{\beta}.K)^{\beta\rightarrow\gamma}\ [N^{\alpha}/x^{\alpha}]$ β has smaller order than α

Observe that it suffices that N is safe.

Tree with substitutions: function symbols $a, f, g, \ldots$; variables $x, y, \ldots$; and explicit substitutions sub_x.

$$eval(sub_x(s,t)) = s[t/x]$$



Theorem[Courcelle & Knapik]: For fixed finite set of variables: $eval$ is MSO-compatible

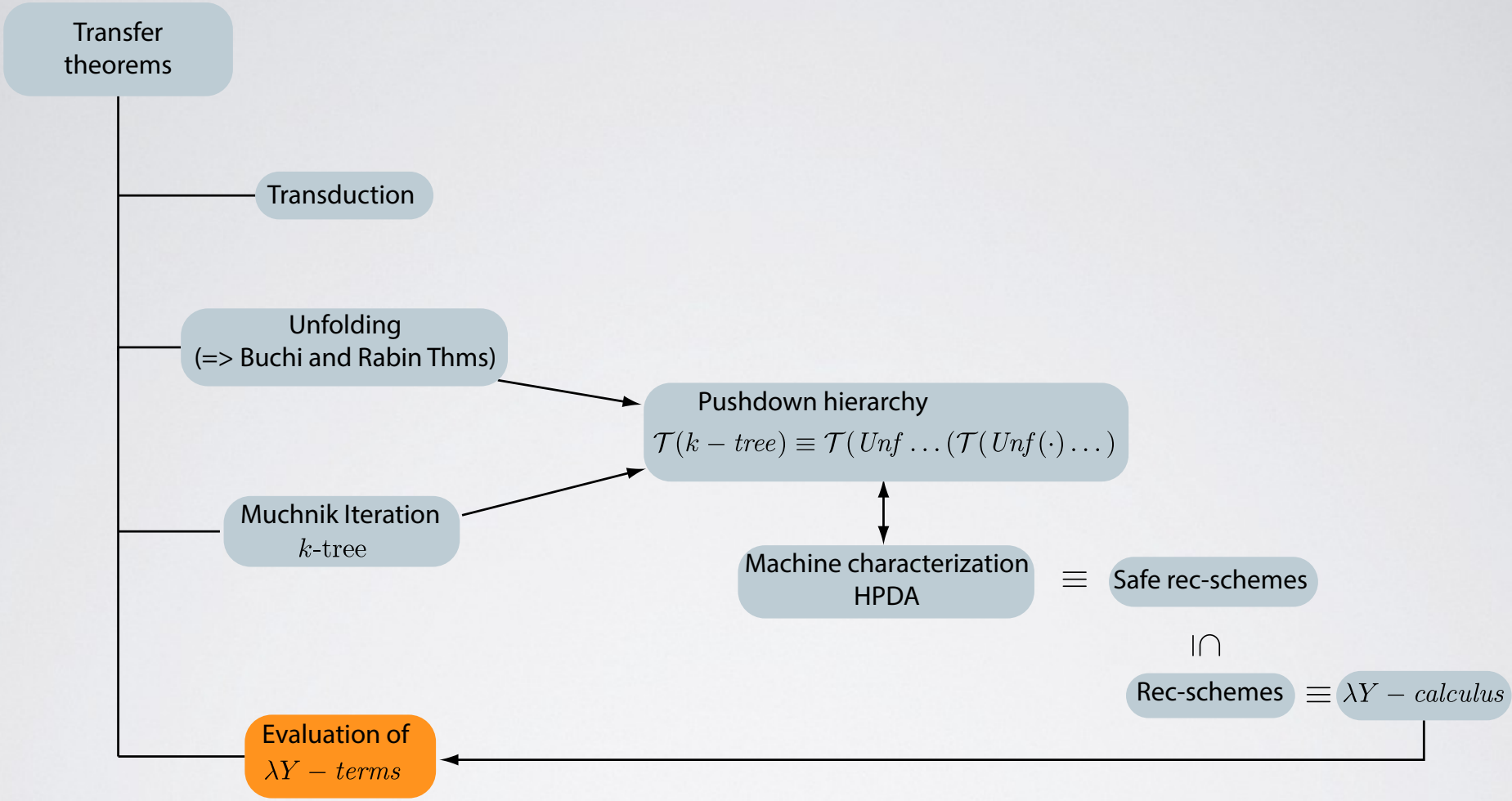
What about schemes that are not safe?

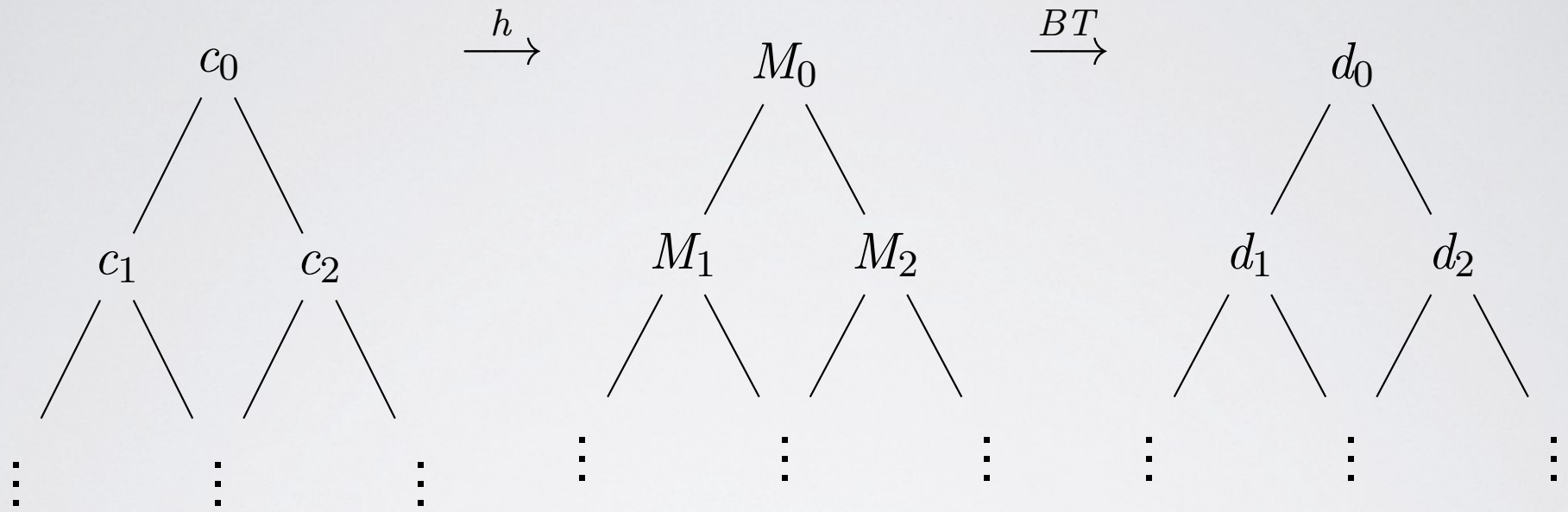
New operations of panic on 2-stack, and then collapse on a higher-order stack. [Urzyczyn, Knapik & Niwiński & Urzyczyn & W., Hague & Murawski & Ong & Serre]

Theorem[Hague & Murawski & Ong & Serre]:
n-th order schemes $\equiv$ unfoldings of n-th order collapse pushdown graphs.

Theorem[Ong]:
MSO theory of the tree generated by a recursive scheme is decidable.

Theorem[Parys]:
Urzyczyn's scheme is not equivalent to a safe scheme.


Transfer theorems
Transduction
Unfolding (=> Buchi and Rabin Thms)
Muchnik Iteration k-tree
Pushdown hierarchy
$\mathcal{T}(k-tree) \equiv \mathcal{T}(Unf \ldots (\mathcal{T}(Unf(\cdot)\ldots)$
Machine characterization HPDA
$\equiv$
Safe rec-schemes
$\cap$
Rec-schemes
$\equiv \lambda Y - calculus$
Evaluation of $\lambda Y - terms$


c_0
c_1
c_2
h
M_0
M_1
M_2
BT
d_0
d_1
d_2

Signature $\Sigma = (B, C)$

- B - a set of base types
- C - a set of constants with types in $Types(B)$.

Terms over Σ defined as usual.

Homomorphism, for two signatures $\Sigma_1 = (B_1, C_1)$, $\Sigma_2 = (B_2, C_2)$, is a function

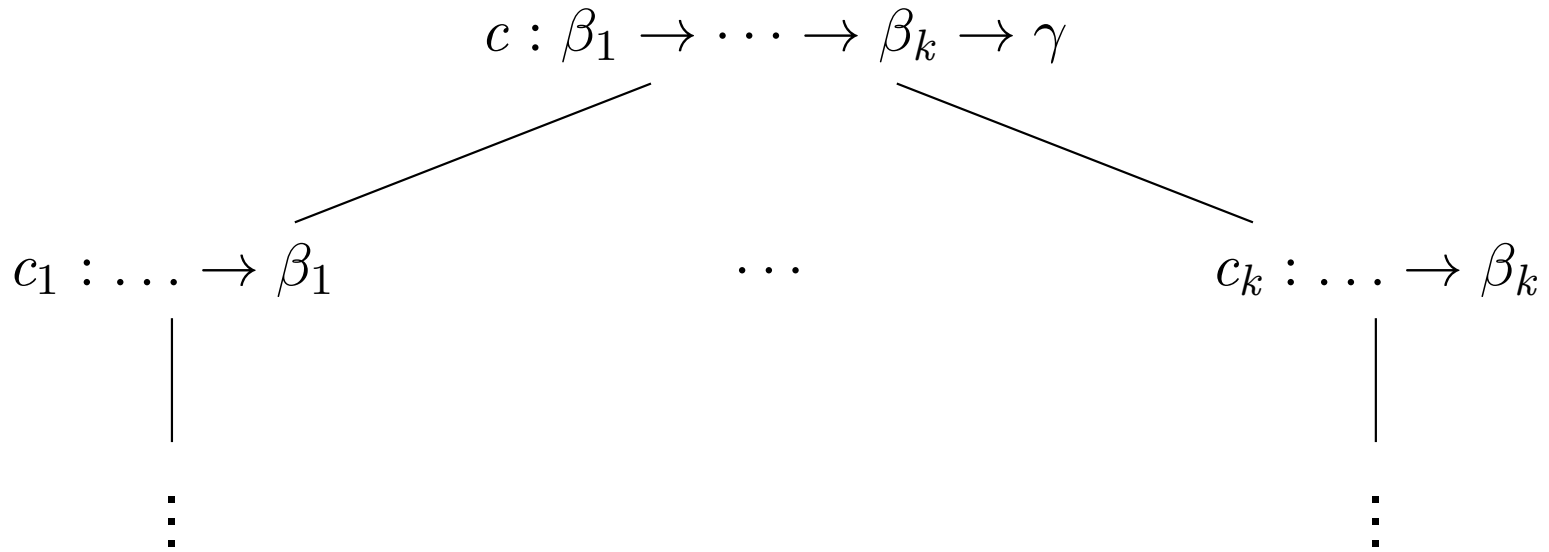
$$h : B_1 \to Types(B_2) \qquad h : C_1 \to Terms(\Sigma_2)$$

with the restriction that $h(c^\alpha)$ is term of type $h(\alpha)$.

First-order signature $\Sigma = (B, C)$:
all constants in C have types of order ≤ 1

$$c : \beta_1 \to \cdots \to \beta_k \to \gamma \qquad \text{with} \qquad \beta_1, \ldots, \beta_k, \gamma \in B.$$

Applicative tree: well typed term (infinite) of a base type constructed only from constants.



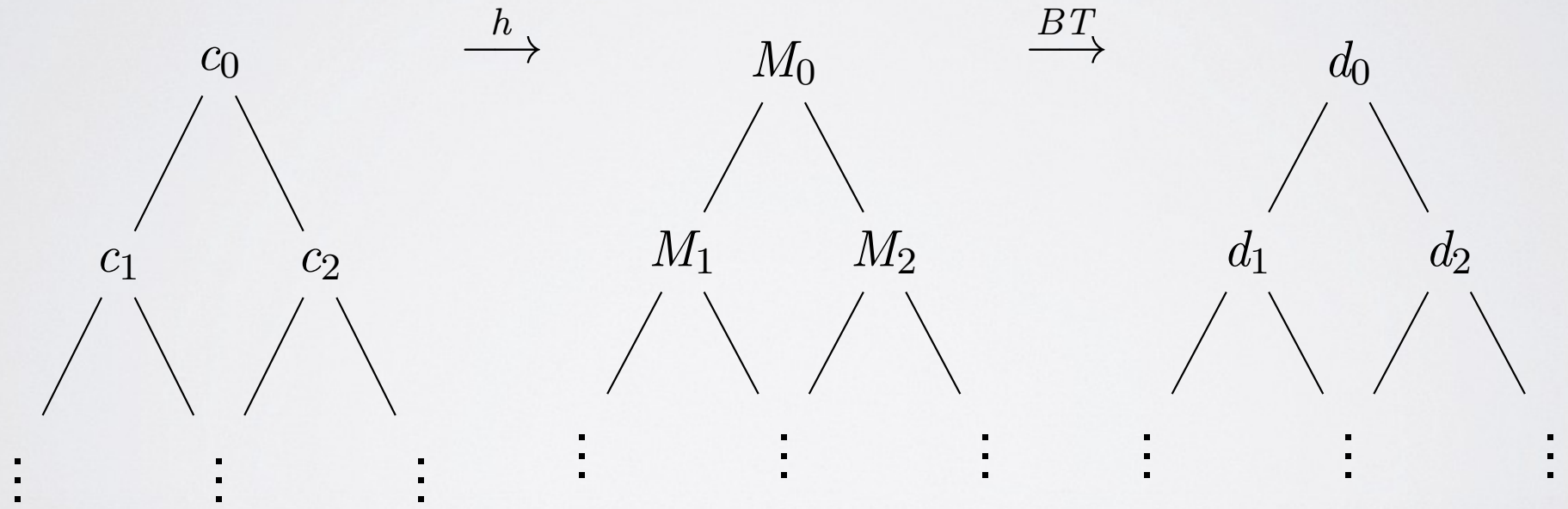
Rem: Applicative trees are just ranked trees so we can talk about their MSO-theories.

First-order signatures $\Sigma_1 = (B_1, C_1)$, $\Sigma_2 = (B_2, C_2)$ and a homomorphism

$$h : B_1 \to Types(B_2) \qquad h : C_1 \to Terms(\Sigma_2)$$

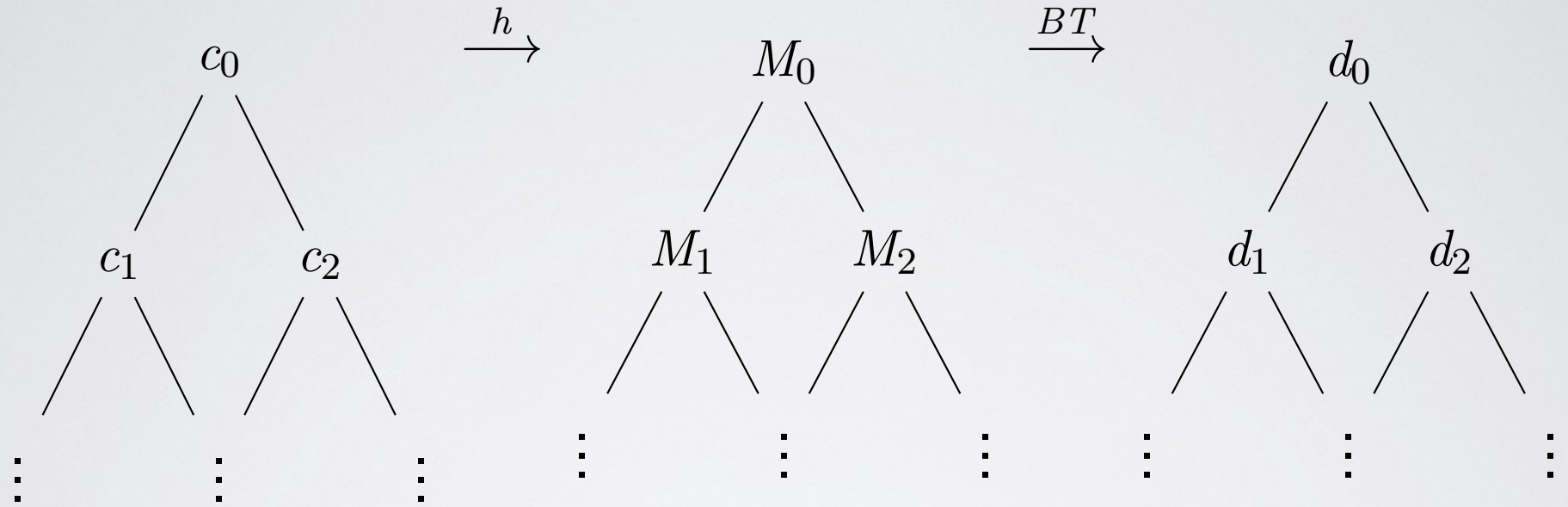
such that $h(\gamma)$ is a base type.

If $t : \gamma$ is an applicative tree over Σ_1 then $BT(h(t))$ is an applicative tree over Σ_2.



Tree operation $t \mapsto BT(h(t))$.

Tree operation $t \mapsto BT(h(t))$



Thm[Salvati & W.]:
Operation $t \mapsto BT(h(t))$ is MSO compatible.

For every φ there is $\widehat{\varphi}$ s.t. for every applicative tree t of type γ:

$$BT(h(t)) \vDash \varphi \qquad \text{iff} \qquad t \vDash \widehat{\varphi}$$

Take an λY-term M and $c : \gamma$. Set $h(c) = M$. We get:

$$BT(h(c)) \vDash \varphi \qquad \text{iff} \qquad c \vDash \widehat{\varphi}$$

This is Ong's theorem: $BT(M)$ has decidable MSO-theory.

Remark:
Every tree in the pushdown hierarchy is $\mathcal{I}(BT(M))$ for some M.

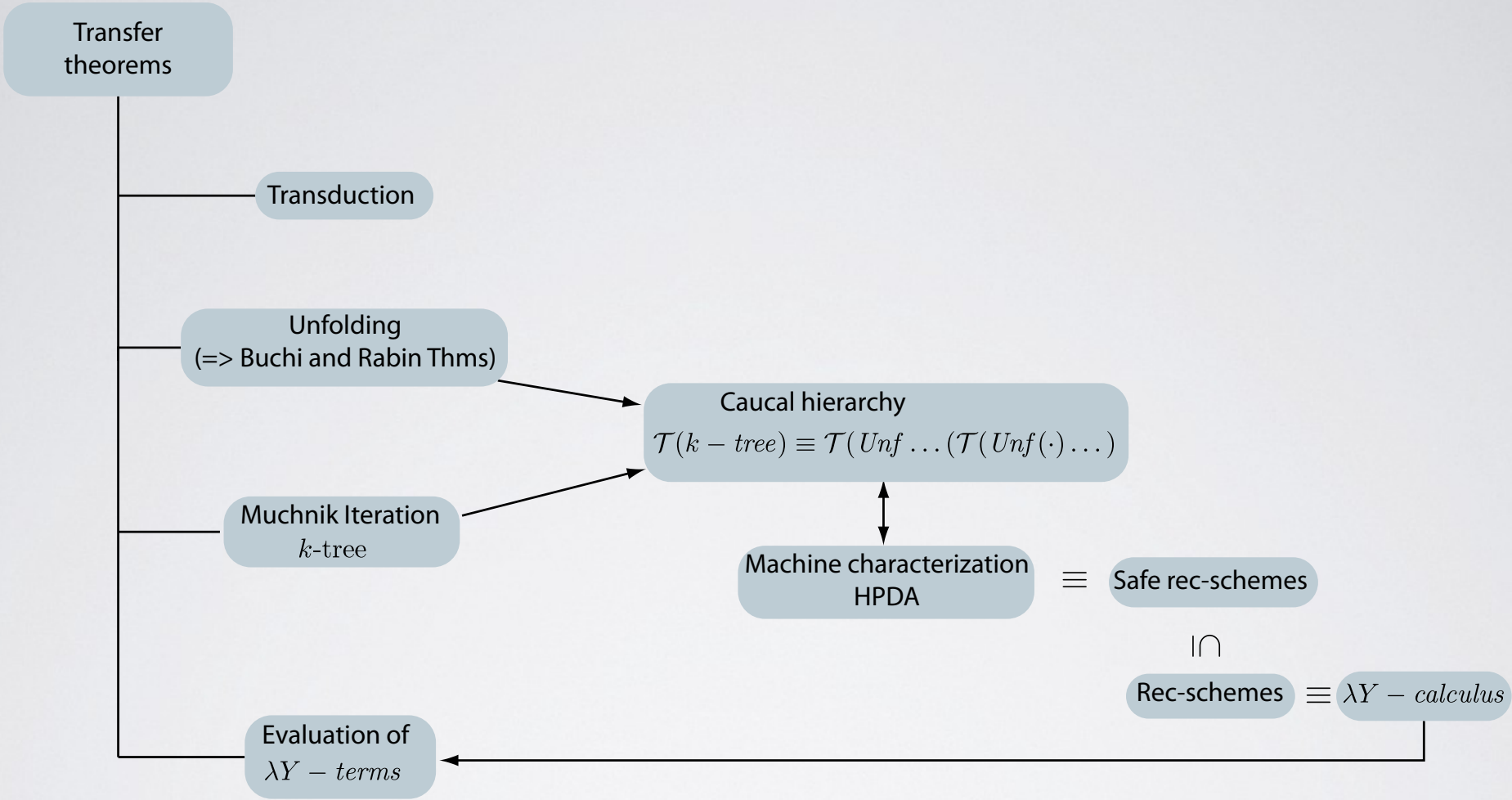
Thm[Parys]:
$BT(M)$ can be outside the pushdown hierarchy.

We have modules $M_1, \ldots, M_k$.
Can we write a program with these modules
whose execution satisfies φ?

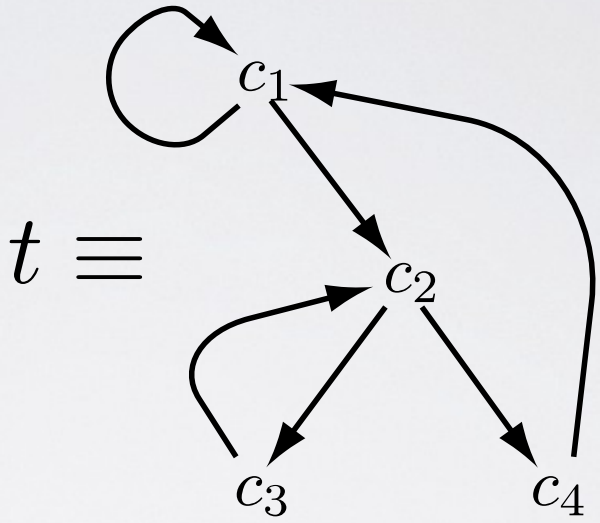
Take homomorphism $h(c_i) = M_i$:

$$BT(h(t)) \models \varphi \qquad \text{iff} \qquad t \models \widehat{\varphi}$$

- Any $t \models \widehat{\varphi}$ gives a program $h(t)$ satisfying φ.
- If $\widehat{\varphi}$ is satisfiable then there is a regular t
- Using the fixpoint combinator we get a finite program for $h(t)$.


Transfer theorems
Transduction
Unfolding (=> Buchi and Rabin Thms)
Muchnik Iteration k-tree
Caucal hierarchy
$\mathcal{T}(k-tree) \equiv \mathcal{T}(Unf \ldots (\mathcal{T}(Unf(\cdot)\ldots)$
Machine characterization HPDA
$\equiv$
Safe rec-schemes
$\cap$
Rec-schemes
$\equiv \lambda Y - calculus$
Evaluation of $\lambda Y - terms$

Scheme of recursive calls

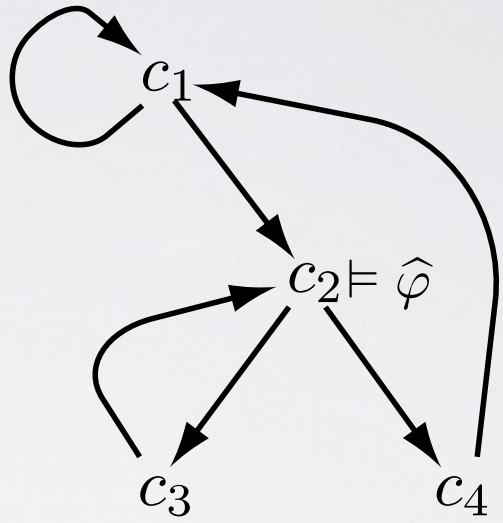


Each call represents a procedure $h(c_i) = M_i$.

Given a property φ we can say at which recursive calls it holds.

Scheme of recursive calls

$$BT(t|_{c_2}) \vDash \varphi$$



Each call represents a procedure $h(c_i) = M_i$.

Given a property φ we can say at which recursive calls it holds.