Transfer Theorems

Igor Walukiewicz
Bordeaux University
Recursion ≡ stacks

\[ F ≡ \lambda x. \text{if } x = 0 \text{ then } 1 \text{ else } F(x - 1) \cdot x. \]

[Courcelle PhD]: Recursive schemes ≡ deterministic pushdown automata.

[Thm Senizergues]: Equivalence of schemes (in terms of trees they generate) is decidable.

[Thm Courcelle]: MSOL theory of trees generated by schemes is decidable.
Recursion $\equiv$ stacks

$F \equiv \lambda x. \text{if } x = 0 \text{ then } 1 \text{ else } F(x - 1) \cdot x.$
Recursion \equiv \text{stacks}

\[
F \equiv \lambda x. \ \text{if} \ x = 0 \ \text{then} \ 1 \ \text{else} \ F(x - 1) \cdot x.
\]

[Courcelle PhD]:
Recursive schemes \equiv \text{deterministic pushdown automata}.
**Recursion ≡ stacks**

\[ F \equiv \lambda x. \text{if } x = 0 \text{ then } 1 \text{ else } F(x - 1) \cdot x. \]

[Courcelle PhD]: Recursive schemes ≡ deterministic pushdown automata.

Thm [Senizergues]: Equivalence of schemes (in terms of trees they generate) is decidable.

Thm [Courcelle]: MSOL theory of trees generated by schemes is decidable.
What about higher-order schemes?

Second-order scheme

\[ \text{Map} \equiv \lambda f. \lambda x. \textbf{if } x = \text{nil} \text{ then nil else } f(\text{hd}(x)) \cdot \text{Map}(f, \text{tl}(x)) \]

Thm [Knapik, Niwiński, Urzyczyn]:
Higher-order pushdown automata \(\equiv\) higher-order safe schemes

Thm [Parys]:
Safety is a true restriction

Here:
On decidability of MSO theory of trees generated by higher-order schemes.
Consider an operation $\mathcal{F}$ on models

**Transfer property for** $\mathcal{F}$

For every $\varphi$ one can effectively construct $\hat{\varphi}$, s.t., for every $M$:

$$\mathcal{F}(M) \models \varphi \quad \text{iff} \quad M \models \hat{\varphi}.$$  

We say in this case that $\mathcal{F}$ is **MSO-compatible**.
Graph with labelled edges: $G = \langle V, \{E_a\}_{a \in \Sigma} \rangle$

Graph with edge labels from $\Sigma$

$\downarrow$

Graph with edge labels form $\Delta$

determined by formulas: $\{\varphi_a(x, y)\}_{a \in \Delta}$
MSO-interpretations are MSO compatible. For every $\varphi$ one can effectively construct $\hat{\varphi}$, s.t., for every $M$:

$$\mathcal{I}(M) \models \varphi \iff M \models \hat{\varphi}.$$ 

$$\hat{\varphi} \equiv \varphi[\varphi_a(x, y) \mapsto E_a(x, y)]$$
$k$-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-transduction is a sequence of copying and MSO interpretations

**Fact:** MSO-transduction is MSO compatible.

\[
\begin{align*}
M_0 & \xrightarrow{\text{copy}} M_1 & & \xrightarrow{\mathcal{I}} M_2 & & \ldots & & \xrightarrow{\text{copy}} M_{k-1} & & \xrightarrow{\mathcal{I}} M_k \\
\varphi_0 & \leftarrow & \varphi_1 & \leftarrow & \varphi_2 & & \ldots & & \leftarrow & \varphi_{k-1} & \leftarrow & \varphi_k
\end{align*}
\]

\[
M_0 \models \varphi_0 \iff M_k \models \varphi_k
\]

**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: $\langle V_U, \{ E^a \} \rangle$ where $V_U =$ paths in $G$ starting from $v_0$.

- $E^a(wv, wvu)$ if $E_a(v, u)$, and $w \in V_U$. 

---

**Diagram:**

- Starting node $v$.
- Edge $v \rightarrow vv$.
- Edge $vv \rightarrow vvv$.
- Path $\cdots$
Unfolding: the tree of all the paths in the graph from a given node.

\[\text{Unfolding}(G, v_0) = \langle V_U, \{E^*_a\} \mid a \in \Sigma \rangle\]

where \(V_U\) are paths in \(G\) starting from \(v_0\) and \(E^*_a(v, u)\) if \(E_a(v, u)\) and \(w \in V_U\).
**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 \( v_0 \)
- \( E_a^*(wv, wvu) \) if \( E_a(v, u), \) and \( w \in V^U. \)
**Theorem** [Courcelle & W., Muchnik]:

Unfolding is MSO-compatible.

For every $\varphi(x)$ there is (effectively) $\widetilde{\varphi}(v_0)$ such that for every graph $G$ and its vertex $v_0$:

$$G \models \widetilde{\varphi}(v_0) \iff Unf(G) \models \varphi(v_0)$$

**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**

\[ St(G) = \langle V^+, \{ E_a^* \}_{a \in \Sigma}, \text{son} \rangle \]

where for \( w \in V^*, u, v \in V \):

- \( \text{son}(w, vw) \),
- \( E_a^*(wu, vw) \) 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

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

- \( E_#(wu, wuu) \) 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.
Some things interpretable in $k$-trees

Interpreting $n(n + 1)/2$ in the iteration of the sequence.

Some other things interpretable in $k$-trees [Fratani & Senizergues]:

- $\langle \mathbb{N}, +1, n\sqrt{n} \rangle$
- $\langle \mathbb{N}, +1, n \log(n) \rangle$
- $\langle \mathbb{N}, +1, n^{k_1}, n^{k_1 k_2}, \ldots, n^{k_1 \ldots k_m} \rangle$
Caucał hierarchy

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

Equivalently:

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

$$
\mathcal{T}(k - \text{tree}) \equiv \mathcal{T}(\text{Unf}(\ldots(\mathcal{T}(\text{Unf}(\text{finite graph})\ldots)))
$$

**Cor:** All graphs in the Caucał hierarchy have decidable MSO-theory.
Caucał hierarchy is infinite

For a function $f : \mathbb{N} \to \mathbb{N}$ we define graph $T_f$:

![Diagram](image)

**Thm** [Engelfriet, Carayol & Wöhrle]:

$T_{\exp_k}$ graph is a $k$-level graph but not $(k - 1)$-level graph.

Let $\exp_\omega(n) = \exp_n(n)$.

**Cor:** $T_{\exp_\omega}$ graph is not in the Caucał hierarchy but has decidable MSO theory.
Transfer theorems

Transduction

Unfolding
(=> Buchi and Rabin Thms)

Muchnik Iteration
$k$-tree

Caucal hierarchy
\[ T(k - \text{tree}) \equiv T(Unf \ldots (T(Unf(\cdot)) \ldots) \]

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 $qa_1 \ldots a_k$
- edges
  
  - $qaw \rightarrow qw$ or
  
  - $qaw \rightarrow qba_w$. 
Configuration graph of a pushdown automaton is interpretable in a tree

**Cor:** It has decidable MSO-theory

**Rem:** Turing Machine graphs may have undecidable MSO-theory.
**2-nd ORDER STACK: EXAMPLE**

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

**New operation of copying the top-most stack:**
\[q[w_1] \ldots [w_i] \rightarrow q[w_1][w_1] \ldots [w_i].\]

<table>
<thead>
<tr>
<th>Transition</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_1[a])</td>
<td>(\rightarrow)</td>
<td>(q_1[aa])</td>
</tr>
<tr>
<td>(q_2[a^k])</td>
<td>(\rightarrow)</td>
<td>(q_2[a^{k-1}][a^k])</td>
</tr>
<tr>
<td>(q_3[a][aa] \ldots [a^k])</td>
<td>(\rightarrow)</td>
<td>(q_3[aa] \ldots [a^k])</td>
</tr>
</tbody>
</table>

A system where all 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 $\equiv$ Caucał 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 Caucał level $k$ are configuration graphs of $k$-th order pushdown automata. (when $\varepsilon$-transitions are contracted).
Transfer theorems

Transduction

Unfolding
(⇒ Buchi and Rabin Thms)

Muchnik Iteration

$k$-tree

Caucal hierarchy

$T(k - \text{tree}) \equiv T(Unf \ldots (T(Unf(\cdot)) \ldots)$

Machine characterization

HPDA

\text{\texttt{\lambda}Y - calculus}
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
Simply typed $\lambda$-calculus with fixpoints

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

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

$$f(gc)d : 0$$

```
       f
      /\  \\
g   d
 /  |  /
/   c  |
```
**β-reduction:** \((\lambda x. M) N =_{\beta} M[N/x]\)

\[(\lambda x. f(gx)d)c \rightarrow_{\beta} f(gc)d\]

\[(\lambda z. z(gc)d)(\lambda xy.y) \rightarrow_{\beta} (\lambda xy.y)(gc)d \rightarrow_{\beta} d\]

Substitution is as in logic: one should avoid variable capture

\[(\lambda h. \lambda x. g(hx))(fx) \rightarrow_{\beta} \lambda y. g(fxy)\]

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

\[f : 0 \rightarrow 0 \rightarrow 0, \quad g, h : 0 \rightarrow 0\]
A Böhm tree of a term $M$:

- We reduce $M$ to head normal form:
  \[ M \Rightarrow^*_{\beta} \lambda \overline{x}.KN_1 \ldots N_i \] with $K$ a variable or a constant.
- $BT(M)$ is:
  \[ \begin{array}{c}
  \lambda x.K \\
  BT(N_1) \quad \cdots \quad BT(N_i)
  \end{array} \]

Böhm tree of $(\lambda y.g(hxy))$ is:

\[ \begin{array}{c}
  \lambda y.g \\
  h \\
  x \quad y
  \end{array} \]
Where are trees?

\[ c : 0, \quad g : 0 \to 0, \quad f : 0 \to 0 \to 0 \]

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

\[
\begin{array}{c}
\text{f} \\
/ \\
N_1 \quad N_2
\end{array} \quad \text{or} \quad
\begin{array}{c}
g \\
/ \\
N_1
\end{array} \quad \text{or} \quad
\begin{array}{c}
c 
\end{array}
\]

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

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

**First order signature:** all constants of order \( \leq 1 \).

**Remark:** For closed \( M : 0 \) over a first order signature \( BT(M) \) is a ranked tree.
We add constants \( Y^{(\alpha \rightarrow \alpha) \rightarrow \alpha} \) and \( \omega^\alpha \), for every type \( \alpha \).

New reduction rule \( YM \rightarrow_\delta M(YM) \).

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

\[
YM \rightarrow_\delta M(YM) \equiv (\lambda x.a x)(YM) \\
\rightarrow_\beta a(YM) \\
\rightarrow_\delta a(M(YM)) \\
\rightarrow_\beta a(a(YM)) \rightarrow \ldots
\]
A Böhm tree of a $\lambda Y$-term $M$ is:
- If $M$ has no head normal form then $\omega^\alpha$.
- Otherwise $\lambda\vec{x}.KN_1 \ldots N_i$ is the head normal form and $BT(M)$ is

$$
\begin{array}{c}
\lambda x.K \\
BT(N_1) \quad \ldots \quad BT(N_i)
\end{array}
$$

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

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

- Transduction

Unfolding
(\Rightarrow \text{Buchi and Rabin Thms})

- Muchnik Iteration
  - $k$-tree

\[ \mathcal{T}(k - \text{tree}) \equiv \mathcal{T}(Unf \ldots (Unf(\ldots )) \ldots ) \]

Caucal hierarchy

Machine characterization
- HPDA
  \[ \equiv \text{Safe rec-schemes} \]

\[ \bigcap \text{Rec-schemes} \equiv \lambda Y - \text{calculus} \]
**Recursive program schemes**

**First example**

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

Another program with the same abstract form:

$Rev \equiv \lambda x. \text{if } x = \text{nil} \text{ then } \text{nil} \text{ else } Rev(tl(x)) \cdot hd(x)$

**Second-order scheme**

$Map \equiv \lambda f. \lambda x. \text{if } x = \text{nil} \text{ then } \text{nil} \text{ else } f(hd(x)) \cdot Map(f, \text{tl}(x))$

**Order of a scheme:** maximal order of a “nonterminal”.
**Semantics**

**Example:**

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

\[ \mathcal{F}(c) \rightarrow_{\beta,\delta} a(\mathcal{F}(bc)) \rightarrow_{\beta,\delta} a(a(\mathcal{F}(b(bc)))) \rightarrow_{\beta,\delta} \ldots \]

**Semantics as a tree of execution**

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

\[ \mathcal{F}(c) = g \]

\[ a \rightarrow g \rightarrow b \rightarrow c \]
Recursion schemes \(\equiv\lambda Y\)-calculus

\[
F_1 = \lambda \vec{x}. M_1 \\
\vdots \\
F_n = \lambda \vec{x}. M_n
\]

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

**Fact**

The tree generated from \(F_n\) is \(BT(T_n)\).

There is also a translation from \(\lambda Y\)-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 $\lambda$-term have orders no smaller than that of the term itself.

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

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

$$(\lambda y^\beta.K)^\beta \rightarrow_\gamma [N^\alpha/x^\alpha] \quad \beta \text{ has smaller order than } \alpha$$
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 operation 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** [Parys]:

Urzyczyn’s scheme is not equivalent to a safe scheme.

**Theorem** [Ong]:

MSO theory of the tree generated by a recursive scheme is decidable.
Transfer theorems

- Transduction
- Unfolding (⇒ Buchi and Rabin Thms)
- Muchnik Iteration

Cauca hierarchy

\[ \mathcal{T}(k\text{-tree}) \equiv \mathcal{T}(\text{Unf} \ldots (\mathcal{T}(\text{Unf}()\ldots )) \]

Machine characterization

HPDA

Safe rec-schemes

\[ \bigcap \text{Rec-schemes} \equiv \lambda Y - calculus \]

Evaluation of λY - terms
**Signature** $\Sigma = (B, C)$
- $B$ - a set of base types
- $C$ - a set of constants with types in $\text{Types}(B)$.

Terms over $\Sigma$ defined as usual.

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

$$h : B_1 \rightarrow \text{Types}(B_2) \quad h : C_1 \rightarrow \text{Terms}(\Sigma_2)$$

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 $\leq 1$

$$c : \beta_1 \to \cdots \to \beta_k \to \gamma$$
with $\beta_1, \ldots, \beta_k, \gamma \in B$.

**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

$$h : B_1 \rightarrow Types(B_2) \quad h : C_1 \rightarrow Terms(\Sigma_2)$$

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

If $t : \gamma$ is an applicative tree over $\Sigma_1$ then $BT(h(t))$ is an applicative tree over $\Sigma_2$.

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

\[ \begin{array}{ccccccc}
  & c_0 & \xrightarrow{h} & M_0 & \xrightarrow{BT} & d_0 \\
  & c_1 & \xrightarrow{h} & M_1 & \xrightarrow{BT} & d_1 \\
  & c_2 & \xrightarrow{h} & M_2 & \xrightarrow{BT} & d_2 \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{array} \]
Tree operation $t \mapsto BT(h(t))$

**Thm** [Salvati & W.]:
Operation $t \mapsto BT(h(t))$ is MSO compatible.
For every $\varphi$ there is $\hat{\varphi}$ s.t. for every applicative tree $t$ of type $\gamma$:

$$BT(h(t)) \models \varphi \quad \text{iff} \quad t \models \hat{\varphi}$$

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

$$BT(h(c)) \models \varphi \quad \text{iff} \quad c \models \hat{\varphi}$$

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

**Remark:** Every tree in a Caucal hierarchy is $I(BT(M))$ for some $M$.

**Thm[Parys]:** $BT(M)$ may be outside Caucal hierarchy.
Scheme of recursive calls

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

Given a property $\varphi$ we can say at which recursive calls it holds.
Scheme of recursive calls

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

Given a property $\varphi$ we can say at which recursive calls it holds.
We have modules $M_1, \ldots, M_k$.
Can we write a program with these modules whose execution satisfies $\varphi$?

Take homomorphism $h(c_i) = M_i$:

$$BT(h(t)) \models \varphi \iff t \models \hat{\varphi}$$

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

Transduction

Unfolding
(⇒ Buchi and Rabin Thms)

Muchnik Iteration

Caucal hierarchy

\[ \mathcal{T}(k - \text{tree}) \equiv \mathcal{T}(\text{Unf} \cdots \mathcal{T}(\text{Unf}(\cdots)) \cdots) \]

Machine characterization

HPDA

Evaluation of

\( \lambda Y - \text{terms} \)

Safe rec-schemes

Rec-schemes

\( \equiv \lambda Y - \text{calculus} \)