Recursive Schemes, Krivine Machines, and Monadic Logic

Igor Walukiewicz
Bordeaux University

Joint work with Sylvain Salvati
**PCF (Programming Computable Functions)**

\[
\text{search} \equiv \lambda p : \text{nat} \rightarrow \text{bool}.
\]

\[
\text{letrec } f(x : \text{nat}) : \text{nat} = \text{if } (p x) \text{ then } x \text{ else } f(x + 1) \text{ in } f0
\]

- Proposed by Scott (1969)
- Mitchell "Foundations for Programming Languages" (1996):
  
  *Designed to be easily analyzed, rather than practical language for writing programs. However with some syntactic sugar it is possible to write many functional programs in a comfortable style.*

- PCF has been in the center of interest of semantics
  *"sequentially computable functional", parallel OR, full abstraction.*
Finitary PCF: base types are finite.

$$\text{search} \equiv \lambda p : "\text{nat}" \to \text{bool}.$$  

$$\text{letrec } f(x : "\text{nat}" ) : "\text{nat}" = \text{if } (px) \text{ then } x \text{ else } f(x + 1) \text{ in } f0$$

- [Statman’04]: $\beta\delta$-equality on terms is undecidable.
- [Loader’96]: There is no recursive fully-abstract model

Finitary PCF $\equiv \lambda Y$-calculus
simply-typed $\lambda$ calculus with fixpoint operators.
\[
\text{map}(f, l) \equiv \begin{cases} 
\text{nil} & \text{if } l = \text{nil} \\
\text{cons}(f(\text{head}(l)), \text{map}(f, \text{tail}(l))) & \text{else}
\end{cases}
\]

\[
\text{map}(f, (a, b, c)) = (f(a), f(b), f(c))
\]
\[ \text{map}(f, l) \equiv \begin{cases} \text{nil} & \text{if } l = \text{nil} \\ \text{cons}(f(\text{head}(l)), \text{map}(f, \text{tail}(l))) & \text{else} \end{cases} \]
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 $f$
Resource usage for functional programs

[Kobayashi’09]

let rec g x = if b then close(x)
           else read(x); g(x) in
let r = open_in "foo" in g(r)

One can verify if usage patterns are correct.
While-programs

\[ x := e | \text{if } x = 0 \text{ then } I_1 \text{ else } I_2 | \text{while } x > 0 \text{ do } I \]

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.
Recursion $\equiv$ stacks

Thm [Courcelle PhD]: 1st order recursive schemes $\equiv$ deterministic pushdown automata.

Thm [Senizergues]: Equivalence of 1st order schemes (in terms of trees they generate) is decidable.

Thm [Courcelle]: MSOL theory of trees generated by 1st order schemes is decidable.
Recursion \equiv \text{stacks}

\[ F \equiv \lambda x. \begin{cases} 1 & \text{if } x = 0 \\ F(x - 1) \cdot x & \text{else} \end{cases} \]
Recursion $\equiv$ stacks

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

**Thm** [Courcelle PhD]:

1-st order recursive schemes $\equiv$ deterministic pushdown automata.
Recursion ≡ stacks

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

**Thm** [Courcelle PhD]:
1-st order recursive schemes ≡ deterministic pushdown automata.

**Thm** [Senizergues]:
Equivalence of 1-st order schemes (in terms of trees they generate) is decidable.

**Thm** [Courcelle]:
MSOL theory of trees generated by 1-st order schemes is decidable.
What about higher-order schemes?

**Second-order scheme**

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

**Thm [Knapik, Niwiński, Urzyczyn]:**

Higher-order **safe** schemes \(\equiv\) higher-order pushdown automata

**Theorem [Hague, Murawski, Ong & Serre]:** \(n\)-th order schemes \(\equiv\) unfoldings of \(n\)-th order collapse pushdown automata.

**Thm [Parys]:**

Safety is a true restriction

**Here:**

On MSO theories of trees generated by higher-order schemes
(These are also the tress generated by programs of finitary PCF).
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
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]
The model-checking problem:
Given $S$ and an MSOL formula $\varphi$ decide if $[S] \models \varphi$.

**Theorem** [Ong]:
This problem is decidable.
Motivation

- Finitary PCF is an important abstraction of functional languages.
- Finitary PCF $\equiv$ schemes $\equiv \lambda Y$-calculus.
- It has been studied by semantics and language communities since 60’ties.
- The “schematological” approach to semantics gives non-trivial insights and without (sometimes) sacrificing decidability.

**Objective**: Understanding trees generated by PCF programs
Preparation

- $\lambda Y$-terms.
- Evaluation.
- Böhm trees.
- MSOL and automata.

$$M \xrightarrow{\text{eval}} BT(M) \in L(A)$$
**Simply typed \(\lambda\)-calculus**

### Types:
- 0 is a type;
- \(\alpha \to \beta\) is a type if \(\alpha, \beta\) are types.

**Eg.** \((0 \to 0) \to 0\)

### Typed constants:
- \(c^\alpha\) for a type \(\alpha\).

### Tree signature:
All constants of types \(0 \to \cdots \to 0 \to 0\).

### Typed terms:
- \(c^\alpha\),
- \(x^\alpha\),
- \((M^\alpha \to \beta N^\alpha)^\beta\),
- \((\lambda x^\alpha.M^\beta)^{\alpha \to \beta}\).
- **Types:** $0 | \alpha \to \beta$
- **Constants:** $c^\alpha$
- **Terms:** $c^\alpha, \ x^\alpha, \ (M^{\alpha\to\beta}N^\alpha)^\beta, \ (\lambda x^\alpha.M^\beta)^{\alpha\to\beta}$.

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

![Diagrams](image)
**β-reduction:** \((\lambda x. M) N \rightarrow^\beta M[N/x]\)

- \((\lambda x.f(gx)d)c \rightarrow^\beta f(gc)d\)
- \((\lambda z. z(gc)d)(\lambda xy.y) \rightarrow^\beta (\lambda xy.y)(gc)d \rightarrow^\beta d\)

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 \(\lambda x.g(fxx)\)

\[f : 0 \rightarrow 0 \rightarrow 0, \quad g, h : 0 \rightarrow 0\]
Result of the computation \( \equiv \) normal form

- \((\lambda x. f(gx)d)c \rightarrow_\beta f(gc)d\)
- \((\lambda z.z(gc)d)(\lambda xy.y) \rightarrow_\beta (\lambda xy.y)(gc)d \rightarrow_\beta d\)
- \((\lambda h. \lambda x.g(hx))(fx) \rightarrow_\beta \lambda y.g(fxy)\)
**Example (QBF)**

- \( \text{tt} = \lambda xy. x \), \( \text{ff} = \lambda xy. y \), They are of type \( 0 \to 0 \to 0 \).
- \( \text{and} = \lambda b_1 b_2. \lambda xy. b_1(b_2xy)y \), \( \text{or} = \lambda b_1 b_2. \lambda xy. b_1x(b_2xy) \),
- \( \text{neg} = \lambda b. \lambda xy. byx \)
- \( \text{All} = \lambda f. \text{and}(f \text{tt})(f \text{ff}) \), \( \text{Exists} = \lambda f. \text{or}(f \text{tt})(f \text{ff}) \).

**QBF to terms**

Every QBF formula \( \alpha \) can be translated to a term \( M_\alpha \):

\[
\forall x. \exists y. x \land \neg y \quad \mapsto \quad \text{All}(\lambda x. \text{Exists}(\lambda y. \text{and} x (\text{neg} y)))
\]

**Fact** For every QBF formula \( \alpha \):

\( \alpha \) is true iff \( M_\alpha \) evaluates to tt.
Let us reduce: \( \text{or (neg tt) tt} \)

\[
\begin{align*}
\lambda b_1 b_2. & \lambda xy. b_1 x (b_2 xy) \\
\lambda xy. & (\text{neg tt}) x (\text{tt} x y)
\end{align*}
\]

We obtain: \( \text{or (neg tt) tt} \rightarrow^* \lambda x y. x \equiv \text{tt} \)
A Böhm tree of a term $M$:

$BT(M)$ is

$\lambda \vec{x}.K$

when $M \rightarrow^*_\beta \lambda \vec{x}.K N_1 \ldots N_i$

$BT(N_1) \ldots BT(N_i)$

Important: If $M : 0$ over tree signature then $BT(M)$ is a ranked tree, the only possible head normal form of $M$ is $aN_1 \ldots N_k$. 

Böhm tree of $(\lambda y. g (h \times y))$ $c$ is

$g$

$\frac{}{h}$

$\frac{x}{c}$
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 $M = (\lambda x.Ax)$

$$YM \rightarrow_\delta M(YM) \equiv (\lambda x.Ax)(YM)$$
$$\rightarrow_\beta A(YM)$$
$$\rightarrow_\delta A(M(YM))$$
$$\rightarrow_\beta A(A(YM)) \rightarrow \ldots$$

What is the result of the computation? $BT(YM) = A^\omega$. 
A Böhm tree of a $\lambda Y$-term $M$ is:

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

For closed terms of type 0 over tree signatures, Böhm tree is a tree.
Digression: Recursion schemes \(\equiv \lambda Y\text{-calculus}\)

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

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

**Fact**

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

There is also a translation from \(\lambda Y\)-terms to schemes.
**Specifying properties of Böhm trees**

**Proviso:** \( \Sigma \) has only constants of types 0 or 0 → 0 → 0 (plus constants \( \Omega^\alpha \), \( Y(\alpha \rightarrow \alpha) \rightarrow \alpha \)).

**Recall:** For tree signature: if \( M \) is a closed term of type 0 then \( BT(M) \) is a ranked tree.

\[
\begin{array}{c}
g \\
| \\
h \\
| \\
\Omega \\
| \\
c
\end{array}
\]

**Monadic second order logic:**

\[
\exists X. \forall y \in X. \exists z \in X. y < z \land a(z)
\]

**Tree automata:**
**Proviso:**

\[ \Sigma = \Sigma_0 \cup \Sigma_2 \text{ with } \Sigma_0 \text{ constants of type } 0 \text{ and } \Sigma_2 \text{ of type } 0 \rightarrow 0 \rightarrow 0. \]

**Tree automaton:**

\[ A = \langle Q, \Sigma \cup \{\Omega\}, q^0 \in Q, \delta_1 : Q \times \Sigma_0 \rightarrow \{\text{false, true}\}, \delta_2 : Q \times \Sigma_2 \rightarrow \mathcal{P}(Q^2) \rangle \]

Run of \( A \):

- \( a_q \):
  - \( (q_0, q_1) \in \delta(q, a) \)
- \( c_{q^0} \)
- \( a_{q^1} \)
- \( c_q \): \( \delta(q, c) = tt \)
- \( \Omega^q \): \( q \in Q \)

**Trivial acceptance condition:** every run is accepting.

**Parity acceptance condition:** max rank on every path is even.
First camp

- $\lambda$-terms $\xrightarrow{\beta\text{-red}}$ Böhm trees (normal form)
- $\lambda Y$-terms $\xrightarrow{\beta\delta\text{-red}}$ Böhm trees with $\Omega$.
- Tree automata running on Böhm trees.
Models
Models

- The meaning of a term is its Böhm tree
- But we can also evaluate terms in models

\[ \text{if } BT(M) = BT(N) \text{ then } [M] = [N] \]
- **Types:** $0 \mid \alpha \to \beta$
- **Constants:** $c^\alpha$
- **Terms:** $c^\alpha, \ x^\alpha, \ (M^\alpha \to ^\beta N^\alpha)^^\beta, \ (\lambda x^\alpha . M^\beta)^{\alpha \to ^\beta}$.

**Model:** $D = \langle \{D^\alpha\}_{\alpha \in \mathcal{T}}, \rho \rangle$

- $D^0$ is a complete lattice;
- $D^{\alpha \to ^\beta}$ monotone functions from $D^\alpha$ to $D^\beta$ ordered coordinatewise;
- $\rho(\Omega^\alpha)$ is the greatest element of $D^\alpha$;
- $\rho(Y^{(\alpha \to ^\alpha) \to ^\alpha})$ is a mapping assigning to a function $f \in D^{\alpha \to ^\alpha}$ its fixpoint.

**GFP model:** when $Y$ assigns greatest fixpoints.

**Finitary model:** when every $D^\alpha$ is finite.
Interpretation of a term $M : \alpha$ in a model $\mathcal{D}$ is an element $[M]_\mathcal{D} \in D^\alpha$.

- $[c]_\mathcal{D}^\nu = \rho(c)$
- $[x^\alpha]_\mathcal{D}^\nu = \nu(x^\alpha)$
- $[MN]_\mathcal{D}^\nu = [M]_\mathcal{D}^\nu[N]_\mathcal{D}^\nu$
- $[\lambda x^\alpha.M]_\mathcal{D}^\nu$ is a function mapping an element $d \in D^\alpha$ to $[M]_\mathcal{D}^\nu[d/x^\alpha]$. (this is a monotone function).

Fact:
For every model $\mathcal{D}$: if $M =_{\beta,\delta} N$ then $[M]^\mathcal{D} = [N]^\mathcal{D}$.

- $\beta$-reduction $(\lambda x. M) N \rightarrow_\beta M[N/x]$
- $\delta$-reduction $Y(M) \rightarrow_\delta M(YM)$.
Example

Take $D^0 = \{0, 1\}$.

Then $D^{0 \rightarrow 0 \rightarrow 0}$ is $\{0, 1\} \rightarrow \{0, 1\} \rightarrow \{0, 1\}$.

$$[\lambda xy. x] = \pi_1 \in D^{0 \rightarrow 0 \rightarrow 0}$$ is the projection on the first component.

$$[\lambda xy. y] = \pi_2 \in D^{0 \rightarrow 0 \rightarrow 0}.$$

For every QBF sentence $\alpha$: $[M_\alpha] = \pi_1$ iff $\alpha$ is true.

Fact For every QBF formula $\alpha$:

$\alpha$ is true iff $M_\alpha$ reduces to $\lambda xy. x$
Thm [Statman’s Weak Completeness Theorem ’82]:
For every $\lambda$-term $M$ there is a finitary model $D_M$ such that for every $\lambda$-term $K$:

$$[M]^{D_M} = [K]^{D_N} \text{ iff } M = \beta K.$$

Thm [Loader’s $\lambda$-definability theorem ’96]:
For every nontrivial finitary model $D$. It is not decidable if a given element $d$ of the model is a denotation of a term.
Interpretation of a term $M : \alpha$ in a model $\mathcal{D}$ is an element $[M]_\mathcal{D} \in D^\alpha$.

- $[c]_\mathcal{D}^\nu = \rho(c)$
- $[x^\alpha]_\mathcal{D}^\nu = \nu(x^\alpha)$
- $[MN]_\mathcal{D}^\nu = [M]_\mathcal{D}^\nu [N]_\mathcal{D}^\nu$
- $[\lambda x^\alpha . M]_\mathcal{D}^\nu$ is a function mapping an element $d \in D^\alpha$ to $[M]_\mathcal{D}^\nu[d/x^\alpha]$.

Fact:
For every model $\mathcal{D}$: if $M =_{\beta,\delta} N$ then $[M]_\mathcal{D} = [N]_\mathcal{D}$.

**Theorem [Barendregt]:** For every finitary GFP-model $\mathcal{D}$:
if $BT(M) \equiv BT(N)$ then $[M]_\mathcal{D} = [N]_\mathcal{D}$. 
$ABT(M)$ is defined by

$$ABT(M) \begin{cases} \lambda x. K & \text{when } M \equiv \lambda x. KN_1 \ldots N_i \\ BT(N_1) & \ldots \\ BT(N_i) & \end{cases}$$

otherwise

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

$\lambda x.a \ x \ (a \ (bx) \ (YF \ (b^2x)))$
**Meanings of Böhm trees**

\[ \lambda x.a \]

\[ x \]

\[ a \]

\[ b^2x \]

\[ a \]

\[ b^2x \]

\[ \omega \]

\[ \lambda x.a \]

\[ x \]

\[ a \]

\[ b^2x \]

\[ a \]

\[ b^2x \]

\[ \omega \]

**Lemma**

\[ BT(M) = \bigsqcup \{ ABT(N) : N =_\beta,\delta M \}; \]

here we are taking syntactic limit over trees.

**Semantics**

\[ [BT(M)]^D = \bigwedge \{ [ABT(N)]^D : N =_\beta,\delta M \} \]

**Theorem [?]**

If \( D \) is a finitary GFP model then: \( [M]^D = [BT(M)]^D \).
**Theorem**

If $D$ is a finitary GFP model then: $[M]^D = [BT(M)]^D$.

**Proof** $[BT(M)] \geq [M]$:

- $[BT(M)]^D = \bigwedge \{[ABT(N)]^D : N = \beta,\delta M\}$.
**Theorem**

If $\mathcal{D}$ is a finitary GFP model then: $\llbracket M \rrbracket^\mathcal{D} = \llbracket BT(M) \rrbracket^\mathcal{D}$.

**Proof** $\llbracket M \rrbracket \geq \llbracket BT(M) \rrbracket$:

- Let $N : \alpha \rightarrow \alpha$ without $Y$:
  - Define $\text{iterate}^i(N)$ to be $N(\ldots(N\Omega^\alpha)\ldots)$.
- Define $\text{iterate}^i(M)$ as the result of repeatedly replacing all $YN$ by $\text{iterate}^i(N)$.

**Obs:** If $\mathcal{D}$ is a finitary GFP model then there is $i$ such that
$\llbracket M \rrbracket^\mathcal{D} = \llbracket \text{iterate}^i(M) \rrbracket^\mathcal{D}$.

$\llbracket M \rrbracket = \llbracket \text{iterate}^i(M) \rrbracket = \llbracket BT(\text{iterate}^i(M)) \rrbracket \geq \llbracket BT(M) \rrbracket$
We have

1. Models $\mathcal{D} = \{D^\alpha\}_{\alpha \in \mathcal{T}, \rho}$ interpreting fixpoint operators.
2. Models are capable of talking about Böhm trees:

$$[[M]^\mathcal{D} = [[BT(M)]^\mathcal{D}$$

We want

- A model $\mathcal{D}_A$ such that $[[M]^{\mathcal{D}_A}$ tells us if $BT(M)$ is accepted by $A$. 
\[ A = \langle Q, \Sigma \cup \{\Omega\}, q^0 \in Q, \]
\[ \delta_1 : Q \times \Sigma_0 \rightarrow \{false, true\}, \delta_2 : Q \times \Sigma_2 \rightarrow \mathcal{P}(Q^2) \rangle \]

**TAC** (trivial acceptance condition) : all runs are accepting.

**Model** \( D_A \):
- \( D^0 = \mathcal{P}(Q) \).
- If \( c : 0 \) then \([c] = \{q : \delta_1(q, c) = true\} \). \([\Omega] = Q \)
- If \( a : 0^2 \rightarrow 0 \) then \([a] \) is a function that for \((S_0, S_1) \in \mathcal{P}(Q)^2 \) returns \( \{q : \delta_2(q, a) \in S_0 \times S_1\} \)

**Theorem**
For every closed term \( M \) of type 0:
\[ BT(M) \in L(A) \iff q_0 \in [M]^{D_A} \]
If $BT(M) \in L(\mathcal{A})$ then $q_0 \in [M]^\mathcal{D}_\mathcal{A}$

Take a run of $\mathcal{A}$ on $BT(M)$ and show that $q^0 \in [BT(M)]^\mathcal{D}_\mathcal{A} = [M]^\mathcal{D}_\mathcal{A}$.

Recall that $[BT(M)]^\mathcal{D} = \bigwedge\{[ABT(N)]^\mathcal{D} : N =_{\beta,\delta} M\}$

We show: $q^0 \in [ABT(N)]$ for $N =_{\beta,\delta} M$. 

\[ 
\begin{array}{c}
\text{a}^{q_0} \\
\downarrow \\
\text{c}^{q_2} \\
\downarrow \\
\text{b}^{q_4} \\
\downarrow \\
\text{c}^{q_2} \\
\downarrow \\
\cdots \\
\text{a}^{q_3} \\
\downarrow \\
\text{a}^{q_1} \\
\downarrow \\
\text{b}^{q_4} \\
\downarrow \\
\text{c}^{q_2} \\
\downarrow \\
\cdots \\
\text{a}^{q_3} \\
\downarrow \\
\text{a}^{q_0} \\
\downarrow \\
\text{c}^{q_2} \\
\downarrow \\
\cdots \\
\text{a}^{q_3} \\
\downarrow \\
\text{a}^{q_1} \\
\downarrow \\
\text{b}^{q_4} \\
\downarrow \\
\text{c}^{q_2} \\
\downarrow \\
\text{c}^{q_2} \\
\downarrow \\
\Omega^{q_2} \\
\downarrow \\
\Omega^{q_3} \\
\end{array} 
\]
If \( q_0 \in [M] \) then \( BT(M) \in L(A) \)

**Property of the interpretation:**

If \( q \in [a(M_0, M_1)] \) then there is \((q_0, q_1) \in \delta(q, a)\) such that: \( q_0 \in [M_0] \), and \( q_1 \in [M_1] \).
A = \langle Q, \Sigma \cup \{\Omega\}, q^0 \in Q, \delta_1 : Q \times \Sigma_0 \rightarrow \{false, true\}, \delta_2 : Q \times \Sigma_2 \rightarrow \mathcal{P}(Q^2) \rangle

Model \( D_A \):

- \( D^0 = \mathcal{P}(Q) \).
- If \( c : 0 \) then \([c] = \{ q : \delta_1(q, c) = true \}\). \( ([\Omega] = Q) \)
- If \( a : 0^2 \rightarrow 0 \) then \([a] \) is a function that for \((S_0, S_1) \in \mathcal{P}(Q)^2 \) returns

\[ \{ q : \delta_2(q, a) \in S_0 \times S_1 \} \]

**Theorem**

For every closed term \( M \) of type 0:

\[ BT(M) \in L(A) \quad \text{iff} \quad q_0 \in [M]^{D_A} \]
To decide $BT(M) \in L(A)$ it is enough to:

- Construct $\mathcal{D}_A$,
- Calculate $[M]^{\mathcal{D}_A}$.

This works only for TAC conditions. (Simple models $\equiv$ TAC conditions)

We can do $\Omega$-aware TAC, but the climb is rather steep.
Reflective Böhm tree wrt. a model $\mathcal{D}$:

Thm [Broadbent, Carayol, Ong, Serre]: For every finitary model $\mathcal{D}$ and $\lambda Y$-term $M$ there is a $\lambda Y$-term $N$ such that $BT(N) = rBT_{\mathcal{D}}(M)$. 
\[(\alpha \to \beta)^\bullet = \alpha^\bullet \to [\alpha] \to \beta^\bullet \text{ and } \alpha^\bullet = \alpha \text{ when } \alpha \text{ is atomic.}\]

\[
[MN, \nu] = [M, \nu] [N, \nu] [N]^\nu
\]
\[
[x^\alpha, \nu] = x^{\alpha^\bullet}
\]
\[
[Y^{(\alpha \to \alpha) \to \alpha} M, \nu] = Y^{(\alpha^\bullet \to \alpha^\bullet) \to \alpha^\bullet} (\lambda x^{\alpha^\bullet}. [M, \nu] x^{\alpha^\bullet} [YM]^\nu)
\]
\[
[\lambda x^\alpha. M, \nu] = \lambda x_0^\alpha \lambda y_{[\alpha]} \cdot \text{case } y_{[\alpha]} \{ d \to [M, \nu[d/x^\alpha]] \} d \in S_\alpha
\]
\[
[a, \nu] = \lambda x_1^0 \lambda y_1^{[0]} \lambda x_2^0 \lambda y_2^{[0]} \cdot \text{case } y_1^{[0]} \{ d_1 \to \text{ case } y_2^{[0]} \{ d_2 \to a^{\rho(a)d_1 d_2 x_1 x_2} \} d_2 \in S_0 \} d_1 \in S_0
\]
Krivine machines
Krivine machines

- The meaning of a term is its Böhm tree.
- It can be computed with a Krivine machine.
- So now instead of using semantics we use syntax.
Our objective is to decide, for a fixed $\mathcal{A}$,

if for a given $M$: $BT(M) \in L(\mathcal{A})$.

We will:

1. use Krivine machine to compute $BT(M)$,
2. construct a game $\mathcal{K}(\mathcal{A}, M)$ on this computation,
3. reduce it to $G(\mathcal{A}, M)$ that will be a finite game.
1. Krivine machine calculating $BT(M)$

2. Acceptance in terms of a game $\mathcal{K}(A, M)$

3. Reduction of $\mathcal{K}(A, M)$ to $G(A, M)$
Krivine machine

- Closure  \( C ::= (N, \rho) \)
- Environment  \( \rho ::= \emptyset \mid \rho[x \mapsto C] \)

<table>
<thead>
<tr>
<th>Term</th>
<th>Environment</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda z. z(gc)d)(\lambda xy. y))</td>
<td>\emptyset</td>
<td>⊥</td>
</tr>
</tbody>
</table>
**Krivine machine**

- **Closure**  \[ C ::= (N, \rho) \]
- **Environment**  \[ \rho ::= \emptyset \mid \rho[x \mapsto C] \]

<table>
<thead>
<tr>
<th>Term</th>
<th>Environment</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda z. z(gc)d)(\lambda xy. y))</td>
<td>(\emptyset)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>(\lambda z. z(gc)d)</td>
<td>(\emptyset)</td>
<td>((\lambda xy. y, \emptyset))</td>
</tr>
</tbody>
</table>
Krivine machine

- **Closure** \( C ::= (N, \rho) \)
- **Environment** \( \rho ::= \emptyset | \rho[x \mapsto C] \)

<table>
<thead>
<tr>
<th>Term</th>
<th>Environment</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda z. z(gc)d)(\lambda xy. y))</td>
<td>(\emptyset)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>(\lambda z. z(gc)d)</td>
<td>(\emptyset)</td>
<td>((\lambda xy. y, \emptyset))</td>
</tr>
<tr>
<td>(z(gc)d)</td>
<td>([z \mapsto (\lambda xy. y, \emptyset)])</td>
<td>(\bot)</td>
</tr>
</tbody>
</table>
# Krivine machine

- **Closure**  
  \[ C ::= (N, \rho) \]

- **Environment**  
  \[ \rho ::= \emptyset \mid \rho[x \mapsto C] \]

<table>
<thead>
<tr>
<th>Term</th>
<th>Environment</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda z. z(gc)d)(\lambda xy. y))</td>
<td>(\emptyset)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>(\lambda z. z(gc)d)</td>
<td>(\emptyset)</td>
<td>((\lambda xy. y, \emptyset))</td>
</tr>
<tr>
<td>(z(gc)d)</td>
<td>([z \mapsto (\lambda xy. y, \emptyset)])</td>
<td>(\bot)</td>
</tr>
</tbody>
</table>

Let \(\rho \equiv [z \mapsto (\lambda xy. y, \emptyset)]\)
Krivine machine

- **Closure** \( C ::= (N, \rho) \)
- **Environment** \( \rho ::= \emptyset \mid \rho[x \mapsto C] \)

<table>
<thead>
<tr>
<th>Term</th>
<th>Environment</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda z. z(gc)d)(\lambda xy. y))</td>
<td>\emptyset</td>
<td>\perp</td>
</tr>
<tr>
<td>((\lambda z. z(gc)d))</td>
<td>\emptyset</td>
<td>((\lambda xy. y, \emptyset))</td>
</tr>
<tr>
<td>((g(d)c))</td>
<td>([z \mapsto (\lambda xy. y, \emptyset)])</td>
<td>\perp</td>
</tr>
<tr>
<td>(z)</td>
<td>(\rho)</td>
<td>((g(c), \rho) (d, \rho))</td>
</tr>
</tbody>
</table>

**Let** \( \rho \equiv [z \mapsto (\lambda xy. y, \emptyset)] \)
**Krivine machine**

- **Closure** \( C ::= (N, \rho) \)
- **Environment** \( \rho ::= \emptyset \mid \rho[x \mapsto C] \)

<table>
<thead>
<tr>
<th>Term</th>
<th>Environment</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda z. z(gc)d)(\lambda xy. y))</td>
<td>(\emptyset)</td>
<td>(\perp)</td>
</tr>
<tr>
<td>(\lambda z. z(gc)d)</td>
<td>(\emptyset)</td>
<td>((\lambda xy. y, \emptyset))</td>
</tr>
<tr>
<td>(z(gc)d)</td>
<td>([z \mapsto (\lambda xy. y, \emptyset)])</td>
<td>(\perp)</td>
</tr>
</tbody>
</table>

Let \(\rho \equiv [z \mapsto (\lambda xy. y, \emptyset)]\)

<table>
<thead>
<tr>
<th>Term</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>(z)</td>
<td>(\rho)</td>
</tr>
<tr>
<td>(\lambda xy. y)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

\( (gc, \rho) \ (d, \rho) \)
## Krivine Machine

- **Closure** \( C ::= (N, \rho) \)
- **Environment** \( \rho ::= \emptyset \mid \rho[x \mapsto C] \)

<table>
<thead>
<tr>
<th>Term</th>
<th>Environment</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (\lambda z. z(gc)d)(\lambda xy. y) )</td>
<td>( \emptyset )</td>
<td>( \bot )</td>
</tr>
<tr>
<td>( \lambda z. z(gc)d )</td>
<td>( \emptyset )</td>
<td>( (\lambda xy. y, \emptyset) )</td>
</tr>
<tr>
<td>( z(gc)d )</td>
<td>([z \mapsto (\lambda xy. y, \emptyset)])</td>
<td>( \bot )</td>
</tr>
<tr>
<td></td>
<td>Let ( \rho \equiv [z \mapsto (\lambda xy. y, \emptyset)] )</td>
<td></td>
</tr>
<tr>
<td>( z )</td>
<td>( \rho )</td>
<td>( (gc, \rho) (d, \rho) )</td>
</tr>
<tr>
<td>( \lambda xy. y )</td>
<td>( \emptyset )</td>
<td>( (gc, \rho) (d, \rho) )</td>
</tr>
<tr>
<td>( y )</td>
<td>([x \mapsto (gc, \rho)][y \mapsto (d, \rho)])</td>
<td>( \bot )</td>
</tr>
</tbody>
</table>
### Krivine Machine

- **Closure**  \[ C ::= (N, \rho) \]
- **Environment**  \[ \rho ::= \emptyset \mid \rho[x \mapsto C] \]

<table>
<thead>
<tr>
<th>Term</th>
<th>Environment</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda z. z(gc)d)(\lambda xy. y))</td>
<td>(\emptyset)</td>
<td>(\perp)</td>
</tr>
<tr>
<td>(\lambda z. z(gc)d)</td>
<td>(\emptyset)</td>
<td>((\lambda xy. y, \emptyset))</td>
</tr>
<tr>
<td>(z(gc)d)</td>
<td>([z \mapsto (\lambda xy. y, \emptyset)])</td>
<td>(\perp)</td>
</tr>
<tr>
<td><strong>Let</strong> (\rho \equiv [z \mapsto (\lambda xy. y, \emptyset)])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(z)</td>
<td>(\rho)</td>
<td>((gc, \rho) (d, \rho))</td>
</tr>
<tr>
<td>(\lambda xy. y)</td>
<td>(\emptyset)</td>
<td>((gc, \rho) (d, \rho))</td>
</tr>
<tr>
<td>(y)</td>
<td>([x \mapsto (gc, \rho)][y \mapsto (d, \rho)])</td>
<td>(\perp)</td>
</tr>
<tr>
<td>(d)</td>
<td>(\rho)</td>
<td>(\perp)</td>
</tr>
</tbody>
</table>
A configuration of a Krivine machine is a triple \((N, \rho, S)\) where:

- \(N\) is a term (a subterm of \(M\));
- \(\rho\) is an environment defined for all free variables of \(N\);
- \(S\) is a stack \(C_1 \ldots C_k\), where \(k\) and the types of the closures are determined by the type of \(N\): the type of \(C_i\) is \(\alpha_i\) where the type of \(N\) is \(\alpha_1 \rightarrow \cdots \rightarrow \alpha_k \rightarrow 0\).

A configuration \((N, \rho, S)\) represents a term:

\[
E((N, \rho, S)) = E(N, \rho)E(C_1) \ldots E(C_n)
\]

Example:

\((z^{0 \rightarrow 0 \rightarrow 0}, \rho, (gc, \rho)(d, \rho))\) with \(\rho \equiv [z \mapsto (\lambda xy. y, \emptyset)]\) gives

\((\lambda xy. y) \ (gc) \ d\)
**Krivine machine**

\[(\lambda x.N, \rho, (K, \rho')S) \rightarrow (N, \rho[x \mapsto (K, \rho')], S)\]

\[(YN, \rho, S) \rightarrow (N(YN), \rho, S)\]

\[(NK, \rho, S) \rightarrow (N, \rho, (K, \rho)S)\]

\[(x, \rho, S) \rightarrow (N, \rho', S) \quad \text{where} \quad (N, \rho') = \rho(x)\]

**Lemma:** Term \(E(N, \rho, \bot)\) has a head normal form iff Krivine machine reduces \((N, \rho, \bot)\) to a \((b(N_1, N_2), \rho', \bot)\) for some constant \(b \neq \Omega\).

**Lemma:** All the terms appearing in configurations of the Krivine machine during the computation from \((M, \emptyset, \bot)\) are subterms of \(M\).
The Böhm tree of

\[ Y(\lambda F. \lambda x. a \ x \ (F(bx))) \ c : 0 \]

is

\[ (\lambda x. \ a \ x \ (YF(bx)))c \]

\[ ac(YF(bc)) \]

\[ c \quad YF(bc) \]

\[ a(bc)YF(b^2c) \]

\[ b \quad YF(b^2c) \]

\[ c \]
**Computing Böhm tree**

**Lemma:** Term $E(N, \rho, \perp)$ has a head normal form iff Krivine machine reduces $(N, \rho, \perp)$ to a $(b(N_1, N_2), \rho', \perp)$ for some constant $b \neq \Omega$.

\[ K_{\text{tree}}(N, \rho, \perp) = BT(N\rho) \]
**Proposition:** For every closed $\lambda Y$-term $M$ of type 0:

\[
BT(M) = Ktree(M, \emptyset, \bot).
\]
1. Krivine machine calculating $BT(M)$

\[(\lambda x. a x (YF(\lambda x)))c\]

\[\alpha(YF(bc))\]

\[c \quad YF(bc)\]

\[\alpha(bc)YF(b^2c)\]

\[b \quad YF(b^2c)\]

\[c \quad \ldots\]

2. Acceptance in terms of a game $\mathcal{K}(A, M)$

3. Reduction of $\mathcal{K}(A, M)$ to $G(A, M)$
**Game for Automaton Acceptance**

**Run of $A$ on $t$**

- $q : b$
- $(q_1, q_2) \in \delta(q, b)$
- $q_1 : c$
- $q_2 : d$

**Acceptance game $G(A, t)$**

- $q : b$
- $(q_1, q_2) : b$
- $(q_1^k, q_2^k) : b$
- for all $(q_1^i, q_2^j) \in \delta(q, \delta)$
- $q_1^i : c$
- $q_2^i : d$
- $q_1^k : c$
- $q_2^k : d$
Eve has a strategy in $G(\mathcal{A}, t)$ iff $t$ is accepted by $\mathcal{A}$. 
Defining $\mathcal{K}(A, M)$

Bohm tree

Krivine machine computing BT

$(N, \rho, \bot)$

$(b(N_1, N_2), \rho', \bot)$

$(N_1, \rho_1, \bot)$

$(N_2, \rho_2, \bot)$

$(c, \ldots)$

$(d, \ldots)$
Defining $\mathcal{K}(A, M)$

Run of the automaton on the Bohm tree

$q : b$

$q_1 : c$

$q_2 : d$

$(q_1, q_2) \in \delta(q, b)$

Run of the automaton on Krivine machine computation

$q : (N, \rho, \perp)$

$q_1 : (N_1, \rho_1, \perp)$

$q_2 : (N_2, \rho_2, \perp)$

$q : (b(N_1, N_2), \rho', \perp)$

$q_1 : (c, \ldots)$

$q_2 : (d, \ldots)$
Defining $\mathcal{K}(A, M)$

Acceptance of the automaton in terms of a game on the Bohm tree

Acceptance of the automaton in terms of a game on Krivine machine computation

$q : (N, \rho, \bot)$

for all $(q^i_1, q^i_2) \in \delta(q, \delta)$
Definition of $\mathcal{K}(A, M)$

$q^0 : (M, \emptyset, \bot)$

$q : (aN_0N_1, \rho, \bot)$ for $(q_1, q_2) \in \delta(q, b)$

$(q_0, q_1) : (aN_0N_1, \rho, \bot)$

$(q_0, q_1) : (aN_0N_1, \rho, \bot)$

$q_0 : (N_0, \rho, \bot)$

$q_1 : (N_1, \rho, \bot)$

$q : (b(N_1, N_2), \rho', \bot)$

$(q_1^1, q_2^1) : (b(N_1, N_2), \rho', \bot)$

$\cdots$

$(q_1^k, q_2^k) : (b(N_1, N_2), \rho', \bot)$
Definition of $\mathcal{K}(A, M)$

$q^0 : (M, \emptyset, \bot)$

$q : (aN_0N_1, \rho, \bot)$

$(q_0, q_1) : (aN_0N_1, \rho, \bot)$

for $(q_1, q_2) \in \delta(q, b)$

$q : (\lambda x. N, \rho, CS)$

$q : (N, \rho[x \mapsto C], S)$

$q : (Y N, \rho, S)$

$q : (N(Y N), \rho, S)$
**Definition of $\mathcal{K}(A, M)$**

$q^0 : (M, \emptyset, \bot)$

- $q : (aN_0N_1, \rho, \bot)$ for $(q_0, q_1) : (aN_0N_1, \rho, \bot)$
  - $q_0 : (N_0, \rho, \bot)$
  - $q_1 : (N_1, \rho, \bot)$

- $q : (\lambda x.N, \rho, CS)$
  - $q : (N, \rho[x \mapsto C], S)$

- $q : (YN, \rho, S')$
  - $q : (N(YN), \rho, S)$

- $q : (NK, \rho, S')$
  - $q : (N, \rho, (v, K, \rho)S)$

- $q' : (x, \rho', S')$
  - $q' : (K, \rho, S)$

where $\rho' = (v, K, \rho)$

Closure is created

Closure is used
where closure is created closure is used

$q : (NK, \rho, S)$
$q : (N, \rho, (v, K, \rho)S)$

$q' : (x, \rho', S)$
$q' : (K, \rho, S)$

$\rho' = (v, K, \rho)$

where $\rho'(x) = (v, K, \rho)$
**Thm:** Eve wins in $\mathcal{K}(A, M)$ iff $A$ accepts $BT(M)$.

**Proposition:** For every closed $\lambda Y$-term $M$ of type $0$: $BT(M) = Ktree(M, \emptyset, \bot)$. 
1. Krivine machine calculating $BT(M)$

\[
(\lambda x. \ a \ x \ (YF(bx)))c \\
\text{with} \\
\text{Computation of the Krivine machine}
\]

2. Acceptance in terms of a game $\mathcal{K}(A, M)$

\[
q : (aN_0N_1, \rho, S) \\
(q_0, q_1) : (aN_0N_1, \rho, S) \\
q_0 : (N_0, \rho, S) \\
q_1 : (N_1, \rho, S) \\
\cdots \text{for all } (q_0, q_1) \in \delta(a, q)
\]

\[
q' : (y^*F([K_i^{K}]\rho, S')) \\
q' : (y^*F([K_i^{K}]\rho, S')) \\
\text{Computation of the Krivine machine}
\]

3. Reduction of $\mathcal{K}(A, M)$ to $G(A, M)$
Decomposition property for a pushdown

\[ q_c \rightarrow q_a \alpha c \quad q_c \rightarrow q_{\text{pop}} \]

This part does not depend on \( c \). Continuation needs only bounded info about the path.

\((q_{c}, R)\)

\((q_{\alpha \alpha c}, R)\)

\((q_{\text{pop}}, R)\)

\((q'_c, R)\)

\((q''_c, R)\)
Reduction to a finite game

\[ qc \mapsto q_a ac \quad qc \mapsto q_{pop} \]

\[ R_a \subseteq Q \]
Reduction to a finite game (with ranks)

\[ qc \mapsto q_a ac \quad qc \mapsto q_{pop} \]

\[ R_a \subseteq Q \times \text{ranks} \]
Residual of type 0 is from $\mathcal{P}(Q \times [d])$.

Residual of type $0 \rightarrow 0$ is from $\mathcal{P}(Q \times [d]) \rightarrow \mathcal{P}(Q \times [d])$. 

From $\mathcal{K}(A, M)$ to $G(A, M)$
\[ G(\mathcal{A}, M) \]

\[
q : (\lambda x. N, \rho, R \cdot S) \rightarrow q : (N, \rho[x \mapsto R], S)
\]

\[
q : (\alpha(N_0, N_1), \rho, \bot) \rightarrow (q_0, q_1) : (\alpha(N_0, N_1), \rho, \bot)
\]

\[
\text{for } (q_0, q_1) \in \delta(q, \alpha)
\]

\[
(q_0, q_1) : (\alpha(N_0, N_1), \rho, \bot) \rightarrow q_i : (N_i, \rho \downarrow_{rk(q_i)}, \bot)
\]

\[
\text{for } i = 0, 1
\]

\[
q : (YN, \rho, S) \rightarrow q : (N(YN), \rho, S)
\]
Eve wins in a position:

- \( q : (x, \rho, S) \) if \( (q, rk(q)) \in \rho(x)(S) \).
Properties of $G(\mathcal{A}, M)$

**Obs:**
For every $N$ there are finitely many nodes in $G(\mathcal{A}, M)$ containing $N$.

**Thm:** Eve wins in $G(\mathcal{A}, M)$ iff Eve wins in $\mathcal{K}(\mathcal{A}, M)$. 

\[
\begin{align*}
q &: (NK, \rho, S) \\
(q, R) &: (NK, \rho, S) \\
q &: (N, \rho, R \cdot S) \\
q' &: (K, \rho, R_1 \cdots R_l) \\
\end{align*}
\]
**Thm:** Eve wins in $G(\mathcal{A}, M)$ iff Eve wins in $K(\mathcal{A}, M)$.

**Proof:**
where closure is created

\[ R(v) = \{(q', r'), \ldots\} \]
Adjusted residual $R(v) \downarrow_r$

$R(v) = \{(q_1, 1), (q_2, 2)\}$

$R(v) \downarrow_2 = \{(q_2, 1), (q_2, 2)\}$
Adjusted residual $R(v) \upharpoonright_r$

\[ R(v) = \{(q_1, 1), (q_2, 2), (q_3, 3)\} \]

\[ R(v) \upharpoonright_2 = \{(q_2, 1), (q_2, 2), (q_3, 3)\} \]
Adjusted residual $R(v) \downarrow_r$

\[ R(v) = \{(q_1, 1), (q_2, 2), (q_3, 3)\} \]

\[ R(v) \downarrow_2 = \{(q_2, 1), (q_2, 2), (q_3, 3)\} \]

**Notation:** \( \text{res}(v, v_1) = R(v) \downarrow_{\max(v, v_1)} \)

- \( \text{res}(C, v_1) = R(v) \downarrow_{\max(v, v_1)} \) where \( C = (v, L, \rho) \),
- \( \text{res}(\rho, v_1) = \rho_1 \) such that \( \rho_1(x) = \text{res}(\rho(x), v_1) \),
- \( \text{res}(S, v_1) = R_1 \ldots R_k \) where \( R_i = \text{res}(C_i, v_1) \), and \( S = C_1 \ldots C_k \).
When $K : 0 \to 0$

Residual $R(v)(R(w) \mid_{r''}) = \{(q', r'), \ldots\}$ where $r'' = \max(w, v')$
Transferring Eve’s strategy from $\mathcal{K}(\mathcal{A}, M)$ to $G(\mathcal{A}, M)$

\[
\begin{array}{c|c}
G(\mathcal{A}, M) & \mathcal{K}(\mathcal{A}, M) \\
\hline
\circ_1 q : (N, \rho_1, S_1) & \circ_2 q : (N, \rho_2, S_2) \\
\hline
q : (\lambda x.N, \rho_1, RS_1) & \rho_1 = \text{res}(\rho_2, v_2) \\
\hline
q : (N, \rho_1[x \mapsto R], S_1) & S_1 = \text{res}(S_2, v_2) \\
\hline
q : (\lambda x.N, \rho_2, CS_2) & R = \text{res}(C, v_2) \\
\hline
q : (N, \rho_2[x \mapsto C], S_2)
\end{array}
\]
Transferring Eve’s strategy from $\mathcal{K}(\mathcal{A}, M)$ to $G(\mathcal{A}, M)$

$G(\mathcal{A}, M)$

$q : (aN_0N_1, \rho_1, \perp)$

$(q_0, q_1) : (aN_0N_1, \rho_1, \perp)$

$q_0 : (N_0, \rho_1, \perp)$

$q_1 : (N_1, \rho_1, \perp)$

$\mathcal{K}(\mathcal{A}, M)$

$q : (aN_0N_1, \rho_2, \perp)$

$(q_0, q_1) : (aN_0N_1, \rho_2, \perp)$

$q_0 : (N_0, \rho_2, \perp)$

$q_1 : (N_1, \rho_2, \perp)$

$\rho_1 = \text{res}(\rho_2, v_2)$

$S_1 = \text{res}(S_2, v_2)$
Transferring Eve’s strategy from $\mathcal{K}(A, M)$ to $G(A, M)$

We want to show $(q, rk(q)) \in \rho_1(x)(S_1)$.

Suppose $\rho_2(x) = (v, K, \rho)$

- $(q, \max(v, v_2)) \in R(v)(\text{res}(S_2, v_2))$ by def.
- $(q, \max(v, v_2)) \in R(v)(\text{res}(S_2, v_2)) \downarrow_{\max(v, v_2)}$ by prop of $\downarrow$
- $(q, rk(q)) \in R(v)(\text{res}(S_2, v_2)) \downarrow_{\max(v, v_2)}$ by prop of $\downarrow$
- $(q, rk(q)) \in \rho_1(x)(S_1)$ by def.
Decomposition property

Closure \( (v, K^0 \rightarrow^0, \rho) \) is replaced by \( R(v) : \mathcal{P}(Q \times [d]) \rightarrow \mathcal{P}(Q \times [d]) \).

We put \( (q', r') \) in \( R(v)(R_L) \).

We use induction on types.
Transferring Eve’s strategy from \( \mathcal{K}(\mathcal{A}, M) \) to \( G(\mathcal{A}, M) \)

\[
\begin{align*}
G(\mathcal{A}, M) & \\
& \circled1 q : (N, \rho_1, S_1) \\
& \quad \downarrow \\
& \quad (q, R(v_2)) : (NK, \rho_1, S_1) \\
& \quad \downarrow \\
& q : (N, \rho_1, R(v_2) \upharpoonright_{rk(q)} S_1) \\
& \quad \downarrow \\
& q' : (K, \rho_1 \downharpoonright_{r'} \mathcal{R}) \\
& \quad \text{for some } (q', r') \in R(v_2)(\mathcal{R}) \\
\end{align*}
\]

\[
\begin{align*}
\mathcal{K}(\mathcal{A}, M) & \\
& \circled2 q : (N, \rho_2, S_2) \\
& \quad \downarrow \\
& \quad \rho_1 = res(\rho_2, v_2) \\
& \quad \downarrow \\
& \quad S_1 = res(S_2, v_2) \\
& \quad \downarrow \\
& q : (N, \rho_2, (v_2, K, \rho_2)S_2) \\
& \quad \downarrow r' \\
& q' : (K, \rho_2, S_2')
\end{align*}
\]
1. Krivine machine calculating $BT(M)$

$$(\lambda x. ax (YF(bx)))c$$

2. Acceptance in terms of a game $\mathcal{K}(A, M)$

$$(q_0, q_1): (aN_0N_1, \rho, S)$$

$$(\cdots)$$

for all $(q_0, q_1) \in \delta(a, q)$

$q_0: (N_0, \rho, S)$

$q_1: (N_1, \rho, S)$

Computation of the Krivine machine

$q': (a^{NK}K_1, \rho', S')$

$q': (a^{NK}K_1, \rho', S')$

3. Reduction of $\mathcal{K}(A, M)$ to $G(A, M)$

$$q: (NK, \rho, S)$$

$q: (N, \rho, (NK, \rho)S)$

$q': (x, \rho', \emptyset)$

where $\rho'(x) = (NK, \rho)$

$q': (K, \rho, \emptyset)$

$q': (K, \rho, \emptyset)$

$q': (x, \rho', \emptyset)$

where $\rho'(x) = R_i$ and $(q', r') \in R$

$q: (N, \rho, RS)$

$q': (K, \rho, RS)$

$q': (K, \rho, \emptyset)$

for all $(q', r') \in R$

\textbf{Thm:} Eve wins in $G(A, M)$ iff Eve wins in $\mathcal{K}(A, M)$.

\textbf{Obs:} $G(A, M)$ is finite.
## Global model checking

### Representing configurations of a Krivine machine

For closures: \( \langle (M, \rho) \rangle \) is

\[
\begin{array}{c}
M \\
\langle \rho(x_1) \rangle \quad \cdots \\
\langle \rho(x_k) \rangle
\end{array}
\]

For configurations: \( \langle (M, \rho, S_1 \ldots S_l) \rangle \) is \( \langle (M, \rho) \rangle, \langle S_1 \rangle, \ldots, \langle S_1 \rangle \).

### Theorem

For every \( M \) and \( A \), the set:

\[
\{ \langle N, \rho, S \rangle : BT(E(N, \rho, S)) \in L(A) \}
\]

is a regular language of finite trees.
Transfer Theorem
\[ M \xrightarrow{eval} BT(M) \]

For all \( \varphi \) exists \( \widehat{\varphi} \) s.t.

\[ M \models \widehat{\varphi} \quad \text{iff} \quad BT(M) \models \varphi \]
$M \xrightarrow{\text{eval}} BT(M)$
$
\hat{\varphi} \leftarrow \varphi$

**Transfer Theorem**

For all $\Sigma, T, X$.
For all $\varphi$ exists $\hat{\varphi}$ s.t. for all $M \in \text{Terms}(\Sigma, T, X)$:

$M \models \hat{\varphi} \iff BT(M) \models \varphi$
**Example: Unfolding**

Graph $\xrightarrow{\text{unfold}}$ Tree

**MSO-compatibility of unfolding**

For all $\Sigma$.

For all $\varphi$ exists $\hat{\varphi}$ s.t. for all $G \in \text{Graph}(\Sigma)$:

$$G \models \hat{\varphi} \iff Unf(G) \models \varphi$$

**Rem:** This theorem implies Rabin’s Theorem.
**Example: Normalizable terms**

**Transfer Theorem:**
For all $\varphi$ exists $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$ :

$$M \models \hat{\varphi} \iff BT(M) \models \varphi$$

- Take $\varphi \equiv "\text{finite tree}"$
- $BT(M) \models \varphi$ iff $M$ has a normal form.

$$M \models \hat{\varphi} \text{ iff } M \text{ has a normal form}$$

So $\{M \in Terms(\Sigma, \mathcal{T}, \mathcal{X}) : M \text{ has a normal form}\}$ is MSOL-definable.
Transfer Theorem

For all \( \Sigma, T, \mathcal{X} \).

For all \( \varphi \) exists \( \hat{\varphi} \) s.t. for all \( M \in \text{Terms}(\Sigma, T, \mathcal{X}) \) :

\[
M \models \hat{\varphi} \iff BT(M) \models \varphi
\]

- \( \Sigma \) is a tree signature
- \( T \) is a finite set of terms
- \( \mathcal{X} \) is a finite set of \( \lambda \)-variables
- \( \text{Terms}(\Sigma, T, \mathcal{X}) \): terms over \( \Sigma \) with
  - all subterms having type in \( T \),
  - all \( \lambda \)-variables from \( \mathcal{X} \).

Note: Theorem works also for infinite \( \lambda Y \)-terms, and unobunded number of \( Y \) variables.
**What it means** \( M \models \hat{\varphi} \)?

\( M \) is represented as a tree \( \text{Graph}(M) \) over the alphabet

\[
T_{\text{alph}}(\Sigma, \mathcal{T}, \mathcal{X}) = \Sigma \cup \{ @^\alpha, Y^\alpha : \alpha \in \mathcal{T} \} \cup \mathcal{X} \cup \{ \lambda^{\alpha \rightarrow \beta} x^\alpha : \alpha \in \mathcal{T} \land \alpha \rightarrow \beta \in \mathcal{T} \land x^\alpha \in \mathcal{X} \}.
\]
Transfer Thm: For all $\varphi$ exists $\hat{\varphi}$ s.t. for all $M \in \text{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$:

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

A sketch of the proof
Transfer Thm: For all $\varphi$ exists $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, T, X)$:

$$M \models \hat{\varphi} \iff BT(M) \models \varphi$$

$\varphi$ \hspace{1cm} $BT(M) \models \varphi$

\downarrow

$AB$ \hspace{1cm} $BT(M) \in L(A)$

\downarrow

Eve wins in $K(A, M)$

$M \models F^{-1}(\gamma_{win})$

\downarrow

$M \models F$

$G(A, M) \models \gamma_{win}$

$G(A, M)$

Eve wins in $G(A, M)$
Consequences of the transfer theorem
Transfer Thm: For all $\varphi$ exists $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, T, \mathcal{X})$:

$$M \models \hat{\varphi} \iff BT(M) \models \varphi$$

**Ong’s Theorem**

It is decidable if for a given finite term $M$ and MSOL formula $\varphi$, $BT(M) \models \varphi$ holds.

**Proof:** Just test $M \models \hat{\varphi}$. 
Transfer Thm: For all $\varphi$ exists $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$:

$$M \models \hat{\varphi} \iff BT(M) \models \varphi$$

The set of normalizing terms is MSOL definable

For a fixed $\mathcal{T}$ and $\mathcal{X}$ there is a formula defining the set of terms $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$ having a normal form.

Proof: Take $\varphi$ defining the set of finite trees and consider $\hat{\varphi}$. 
Digression: why limiting \( \lambda \)-variables

QBF to terms

Every QBF formula \( \alpha \) can be translated to a term \( M_\alpha \):

\[
\forall x. \exists y. x \land \neg y \quad \leftrightarrow \quad All(\lambda x. \exists y. \text{and } x (\text{not } y))
\]

\( \alpha \) is true iff \( BT(M_\alpha) \) is the term \( true \)

Take \( \varphi \) saying that the tree consists only of the root labeled \( true \). Consider \( \hat{\varphi} \).

\[
M_\alpha \models \hat{\varphi} \quad \text{iff} \quad \alpha \text{ is true.}
\]

If we could construct \( \hat{\varphi} \) without limiting \( \lambda \) then we get collapse of the polynomial hierarchy.
Matching with restricted no of variables

For a fixed $\mathcal{X}$. Given $M$ and $K$ (without fixpoints) decide if there is a substitution $\sigma$ such that

$$M\sigma \equiv_\beta K$$

Substitution $\sigma$ can use only terms from $\text{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$.

Proof:

- Let $shape(N)$ be MSOL formula defining the set of terms in $\text{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ that can be obtained from $N$ by substitutions.
- Let $\varphi \equiv shape(K)$.
- There is desired $\sigma$ iff the formula $shape(M) \land \hat{\varphi}$ is satisfiable.

If there is a solution then there is a finite one.
SYNTHESIS FROM MODULES

Given finite \( \lambda Y \)-terms \( M_1, \ldots, M_k \) and \( \varphi \). Decide if one can construct a \( \lambda Y \) term \( K \) from these terms such that \( BT(K) \models \varphi \).

Proof:

- The candidate term \( K \) can be described as having the form \( (\lambda x_1 \ldots x_k. \ N) M_1, \ldots, M_k \) for some term \( N \) without constants and \( \lambda \)-abstractions.
- Let \( \psi \) be a formula defining terms of this form.
- There is a solution iff the formula \( \psi \land \hat{\varphi} \) is satisfiable.

Every model of \( \psi \land \hat{\varphi} \) gives a solution.

If there is a solution then there is a regular one, hence a finite one thanks to the presence of \( Y \).
Two ways looking at it.

Studding new properties of evaluation in simple types.

Bringing verification to a new ground.
In the beginner’s mind there are many possibilities, in the expert’s mind there are few.
Decomposition property for a pushdown:

\[ \text{qc} \rightarrow q_\alpha ac \quad qc \rightarrow q_{\text{pop}} \]

This part does not depend on c.
Continuation needs only bounded info about the path.
\[ M \xrightarrow{\text{eval}} BT(M) \subseteq L(A) \]

- Understanding \( BT(M) \subseteq L(A) \) in terms of models

\[ \llbracket M \rrbracket = \llbracket BT(M) \rrbracket \]

- Understanding \( BT(M) \subseteq L(A) \) in terms of \( K(A, M) \)

\[ K(A, M) \text{ equivalent to } G(A, M) \]

- MSO compatibility of evaluation

\[ M \xrightarrow{\text{eval}} BT(M) \]

\( \hat{\phi} \leftarrow \phi \)
Getting closer to “real” computation.

Transfer theorem covers: Rabin’s theorem, unfolding theorem, pushdown hierarchy, Ong’s theorem, global model-checking,…

The use of old techniques in a new way.