Transfer Theorem

Sylvain Salvati and Igor Walukiewicz
Bordeaux University
\[ M \xrightarrow{eval} BT(M) \]
\[ M \xrightarrow{\text{eval}} \text{BT}(M) \]

**Transfer Theorem**

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

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

**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**

\[ \text{Graph} \xrightarrow{\text{unfold}} \text{Tree} \]

**MSO-compatibility of unfolding**

For all $\Sigma$.

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

\[ G \models \widehat{\varphi} \quad \text{iff} \quad \text{Unf}(G) \models \varphi \]

**Rem:** This theorem implies Rabin’s Theorem.
LIBERTÉ D'ABORD

PLUS DE DÉMAGOGIE VOTEZ R.P.R.

A BAS TOUS POUR LE R.P.R.

IN!
**PCF (Programming Computable Functions)**

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

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

- 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.

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

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

- **[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.
map(f, l) ≡ \textbf{if } l = \texttt{nil} \textbf{ then } \texttt{nil} \\
\textbf{else cons}(f(\texttt{head}(l)), \texttt{map}(f, \texttt{tail}(l)))

\[
\text{map}(f, (a, b, c)) = (f(a), f(b), f(c))
\]
\[
\text{map}(f, l) \equiv \begin{cases} 
\text{if } l = \text{nil} \text{ then nil} \\
\text{else cons}(f(\text{head}(l)), \text{map}(f, \text{tail}(l)))
\end{cases}
\]
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

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

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

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

Thm \([\text{Courcelle PhD}]\)\: 1-st order recursive schemes \(\equiv\) deterministic pushdown automata.

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

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

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

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

**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
\[ M \xrightarrow{eval} BT(M) \]
Transfer Theorem

For all $\varphi$ exists $\hat{\varphi}$ s.t.

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

For all $\Sigma, \mathcal{T}, \mathcal{X}$.

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

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

\[
\text{Graph} \xrightarrow{\text{unfold}} \text{Tree}
\]

**MSO-compatibility of unfolding**

For all \( \Sigma \).

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

\[
G \models \hat{\varphi} \iff \text{Unf}(G) \models \varphi
\]

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

**Transfer Theorem**

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

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

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

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

So $\{M \in Terms(\Sigma, T, X) : M \text{ has a normal form}\}$ is MSOL-definable.
\[ M \xrightarrow{\text{eval}} BT(M) \]

\( \lambda Y \)-terms
Types: 0 is a type, and $\alpha \rightarrow \beta$ is a type if $\alpha, \beta$ types.

Tree signature $\Sigma = \{a, b, \ldots\}$ all constants of type $0 \rightarrow 0 \rightarrow 0$.

Terms

$\Omega^\alpha$, $x^\alpha$, $x^\alpha$, or $c^\alpha$ are in $T_\alpha$.

$\lambda^{\alpha\rightarrow\beta} \mid T_\beta$ is in $T_{\alpha\rightarrow\beta}$.

$Y^\alpha x \mid T_\alpha$ is in $T_\alpha$. 
Example of terms

\[ Yx. \]

\[ \lambda z. \]

\[ @ \]

\[ x \]

\[ z \]

Malformed tree

\[ \lambda x. \]

\[ @ \]

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

Evaluation and \( BT(M) \)
- \( \beta \)-reduction: \((\lambda x.M)N \rightarrow_{\beta} M[x := N]\).
- \( \delta \)-reduction: \(Yx.M \rightarrow_{\delta} M[x := Yx.M]\).

- Head redex: \((\lambda x.P)P_0 \ldots P_n\) or \((Yx.P)P_1 \ldots P_n\)
- Weak head normal form: \(hN_1 \ldots N_k\) with \(h\) a variable or a constant different than \(\Omega\).

**Böhm tree**

Suppose that \(M:0\) over a tree signature.

- if \(M \rightarrow^*_{h} bN_1N_2\) with \(b \neq \Omega\) then \(BT(M) = b\overline{\ \ \ \ \ \ }\) \(BT(N_1)\) \(BT(N_2)\)
- otherwise \(BT(M) = \Omega^0\).
Suppose that \( M : 0 \) over a tree signature.

- if \( M \to^*_h bN_1N_2 \) with \( b \neq \Omega \) then \( BT(M) = \)
  \[
  \begin{array}{c}
  b \\
  \text{BT}(N_1) \quad \text{BT}(N_2)
  \end{array}
  \]
- otherwise \( BT(M) = \Omega^0 \).

**Example 1**

- \( Yx.ax \)
- Every tree generated by a recursive scheme is \( BT(M) \) for some \( M \).
Example 2 (QBF)

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

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 sentence \( \alpha \):

\( \alpha \) is true iff \( M_\alpha \) evaluates to \text{tt}. 
For all $\Sigma, T, X$.
For all $\varphi$ exists $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, T, X)$:

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

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

Note: no limitation on $Y$ variables.
What it means \( M \models \hat{\varphi} \)?

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

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

$$M \models \hat{\varphi} \iff BT(M) \models \varphi$$
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 $M \models \hat{\varphi}$. 
The set of normalizing terms is MSOL definable

For a fixed $T$ and $X$ there is a formula defining the set of terms $M \in \text{Terms}(\Sigma, T, X)$ 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 $M_\alpha$:

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

$\alpha$ 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 $X$ 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

$$M\sigma \beta K$$

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

Proof:

- Let $shape(N)$ be MSOL formula defining the set of terms in $\text{Terms}(\Sigma, T, \mathcal{X})$ 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 $\varphi$ can one construct a $\lambda Y$ term $K$ from these terms such that $BT(M) \models \varphi$.

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

\[
M \models \hat{\varphi} \quad \text{iff} \quad BT(M) \models \varphi
\]
\[ \varphi \quad \text{if} \quad BT(M) \models \varphi \quad \text{iff} \quad M \models F^{-1}(\gamma_{\text{win}}) \quad \text{if} \quad G(A, M) \models \gamma_{\text{win}} \quad \text{iff} \quad \text{Eve wins in } G(A, M) \]
$M$ is in a **canonical form** if no subterm $Yx. N$ of $M$ has free $\lambda$-variables.

\[
Yx. N \mapsto (Yy.\lambda x_1 \ldots \lambda x_n. N[x := yx_1 \ldots x_n]) x_1 \ldots x_n
\]
Krivine machine

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

Initial term

\[
M = (Y x^0 \to^0 . \lambda y^0 . b(x y) y)(Y z^0 . c z z)
\]

A closure

\[
C = (b(x y^0), [y^0 \leftarrow (Y z^0 . c z z, \emptyset)])
\]

Expansion of the closure

\[
E(C) = b(x y^0)[y := E(Y z^0 . c z z, \emptyset)]
\]
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 $\alpha_1 \to \cdots \to \alpha_k \to 0$.

A configuration $(N, \rho, S)$ represents an infinitary term:

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

Example: $(b(x y^0), [y^0 \leftarrow Y z^0.c z z], (y^0, [y^0 \leftarrow Y z^0.c z z]))$
Fix a canonical term $M$. Every $Y$-variable bound at most once in $M$. So $\text{term}(x)$ is the subterm $Yx.N$ of $M$. (It has not free $\lambda$-vars).

**Krivine machine**

$$(\lambda x. N, \rho, (K, \rho')S) \rightarrow (N, \rho[x \mapsto (K, \rho')], S)$$
$$(Yx. N, \rho, S) \rightarrow (N, \rho, S)$$
$$(NK, \rho, S) \rightarrow (N, \rho, (K, \rho)S)$$
$$(x, \rho, S) \rightarrow (N, \rho', S) \quad \text{where } (N, \rho') = \rho(x)$$
$$(x, \rho, S) \rightarrow (\text{term}(x), \emptyset, S)$$

**Lemma**

Term $E(N, \rho, \bot)$ has a head normal form iff Krivine machine reduces $(N, \rho, \bot)$ to a $(b, \rho, S)$ for some constant $b \neq \Omega$.
**Computing Böhm Tree**

**Lemma**

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

For every concrete canonical and closed $\lambda Y$-term $M$ of type $0$:

$$BT(M) = KT(M).$$

All the terms appearing in configurations of the Krivine machine during the computation of $KT(M)$ are subterms of $M$. 
A sketch of the proof

- \( \varphi \) \( \Downarrow \)
  - \( BT(M) \models \varphi \)
    - iff
    - \( A \)
      - \( BT(M) \in L(\mathcal{A}) \)
    - iff
    - Eve wins in \( K(\mathcal{A}, M) \)

- \( M \Downarrow F \)
  - \( M \models F^{-1}(\gamma_{\text{win}}) \)
    - iff
    - \( G(\mathcal{A}, M) \models \gamma_{\text{win}} \)
    - iff
    - Eve wins in \( G(\mathcal{A}, M) \)
Defining $K(A, M)$

Krivine machine computation

$(N, \rho, \perp)$

$(b, \rho', (N_1, \rho_1)(N_2, \rho_2))$

$(N_1, \rho_1, \varepsilon)$

$(d, \ldots)$

$(N_2, \rho_2, \varepsilon)$

The resulting Bohm tree

$b$

$c$

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

Run of the automaton on the Bohm tree

$q_1 : c$

$q_2 : d$

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

Run of the automaton on Krivine machine computation

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

$q_1 : (b, \rho', (N_1, \rho_1)(N_2, \rho_2))$

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

$q_1 : (c, \ldots)$

$q_2 : (d, \ldots)$

$q \ast$
**Defining $\mathcal{K}(A, M)$**

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

For all $(q^i_1, q^i_2) \in \delta(q, \delta)$

- $q_1^i : b$
- $q_2^i : d$
- $q_1^i : c$
- $q_2^i : d$

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

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

$q : (b, \rho', (N_1, \rho_1)(N_2, \rho_2))$

- $(q_1^1, q_2^1) : (b, \rho', (N_1, \rho_1)(N_2, \rho_2))$
- $(q_1^2, q_2^2) : (b, \rho', (N_1, \rho_1)(N_2, \rho_2))$

- $q_1^1 : (N_1, \rho_1, \perp)$
- $q_1^2 : (N_1, \rho_1, \perp)$
- $q_2^1 : (N_2, \rho_2, \perp)$
- $q_2^2 : (N_2, \rho_2, \perp)$
Definition of \( \mathcal{K}(A, M) \)

- The root is \( q^0 : (M, \emptyset, \bot) \).
- A node \( q : (a, \rho, S) \) has a successor \((q_0, q_1) : (a, \rho, S)\) for every \((q_0, q_1) \in \delta(q, a)\).
- A node \((q_0, q_1) : (a, \rho, (v_0, N_0, \rho_0)(v_1, N_1, \rho_1))\) successors \(q_0 : (N_0, \rho_0, \bot)\) and \(q_1 : (N_1, \rho_1, \bot)\).
Definition of $\mathcal{K}(A, M)$

- The root is $q^0 : (M, \emptyset, \bot)$
- A node $q : (a, \rho, S)$ has a successor $(q_0, q_1) : (a, \rho, S)$ for every $(q_0, q_1) \in \delta(q, a)$.
- A node $(q_0, q_1) : (a, \rho, (v_0, N_0, \rho_0)(v_1, N_1, \rho_1))$ successors $q_0 : (N_0, \rho_0, \bot)$ and $q_1 : (N_1, \rho_1, \bot)$.
- A node $q : (\lambda x. N, \rho, CS)$ has a successor $q : (N, \rho[x \mapsto C], S)$.
- A node $q : (Y x. N, \rho, S)$ has a successor $q : (N, \rho, S)$.
- A node $q : (x, \rho, S)$, for $x$ a recursive variable, has a successor $q : (\text{term}(x), \emptyset, S)$. 
**Definition of** $\mathcal{K}(A, M)$

- The root is $q^0 : (M, \emptyset, \bot)$
- A node $q : (a, \rho, S)$ has a successor $(q_0, q_1) : (a, \rho, S)$ for every $(q_0, q_1) \in \delta(q, a)$.
- A node $(q_0, q_1) : (a, \rho, (v_0, N_0, \rho_0)(v_1, N_1, \rho_1))$ successors $q_0 : (N_0, \rho_0, \bot)$ and $q_1 : (N_1, \rho_1, \bot)$.
- A node $q : (\lambda x. N, \rho, CS)$ has a successor $q : (N, \rho[x \mapsto C], S)$.
- A node $q : (Y x. N, \rho, S)$ has a successor $q : (N, \rho, S)$.
- A node $q : (x, \rho, S)$, for $x$ a recursive variable, has a successor $q : (\text{term}(x), \emptyset, S)$.
- A node $v$ labeled $q : (NK, \rho, S)$ has a unique successor labeled $q : (N, \rho, (v, K, \rho)S)$. $v$-closure is created.
- A node $v$ labeled $q : (x, \rho, S)$, for $x$ a $\lambda$-variable and $\rho(x) = (v', N, \rho')$, has a unique successor labeled $q : (N, \rho', S)$. Node $v$ uses a $v'$-closure.
A node $v$ labeled $q : (NK, \rho, S)$ has a unique successor labeled $q : (N, \rho, (v, K, \rho)S)$. $v$-closure is created.

A node $v$ labeled $q : (x, \rho, S')$, for $x$ a $\lambda$-variable and $\rho(x) = (v', N, \rho')$, has a unique successor labeled $q : (N, \rho', S)$. Node $v$ uses a $v'$-closure.

**Thm:** Eve wins in $\mathcal{K}(A, M)$ iff $A$ accepts $BT(M)$. 
A sketch of the proof

<table>
<thead>
<tr>
<th>φ</th>
<th>$BT(M) \models \varphi$</th>
<th>$M \models F^{-1}(\gamma_{win})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>iff</td>
<td>iff</td>
</tr>
<tr>
<td>$A$</td>
<td>$BT(M) \in \mathcal{L}^2$</td>
<td>$G(A, M) \models \gamma_{win}$</td>
</tr>
<tr>
<td></td>
<td>iff</td>
<td>iff</td>
</tr>
<tr>
<td></td>
<td>Eve wins in $\mathcal{K}(A, M)$</td>
<td>Eve wins in $G(A, M)$</td>
</tr>
</tbody>
</table>

$M \downarrow F$
From $\mathcal{K}(A, M)$ to $G(A, M)$

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

\[ q : (N, \rho, (v, K, \rho) S) \]

\[ q' : (x, \rho', \emptyset) \] where $\rho'(x) = (v, K, \rho)$

\[ q' : (K, \rho, \emptyset) \]
\( G(\mathcal{A}, M) \)

\[
q : (\lambda x. N, \rho, R \cdot S) \rightarrow q : (N, \rho[x \mapsto R], S) \\
q : (a, \rho, R_0 R_1) \rightarrow (q_0, q_1) : (a, \rho, R_0 R_1) \\
q : (Y x. N, \rho, S) \rightarrow q : (N, \rho, S) \\
q : (x, \rho, S) \rightarrow q : (\text{term}(x), \rho, S)
\]

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

\( x \) a recursion variable
Eve wins in a position:

- \( q : (x, \rho, S) \)
  - if \( (q, rk(q)) \in R_x(R_1, \ldots, R_k) \); where \( \rho(x) = R_x \) and \( S = R_1 \cdots R_k \).

- \( (q_0, q_1) : (a, \rho, R_0 R_1) \)
  - if \( (q_0, rk(q_0)) \in R_0 \downarrow_{rk(q_0)} \) and \( (q_1, rk(q_1)) \in R_1 \downarrow_{rk(q_1)} \).
**Properties of** $G(\mathcal{A}, M)$

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

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

Prop: \( G(\mathcal{A}, M) \) is definable from in \( \text{Graph}(M) \) by means of MSOL-transduction.
Transfer Theorem

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

\[ M \models \varphi \iff BT(M) \models \varphi \]