Separating Terms through Multi Types

Adrienne Lancelot

Inria & LIX, École Polytechnique
IRIF, Université Paris Cité & CNRS

July 9th 2024 – ITRS
Are these program equivalent?

When studying program equivalence, one searches for any small hint that programs are not equivalent:

\( \lambda x. x + x \) and \( \lambda x. x \times x \) can be distinguished by the input 1 (but not by the input 2)

\( \lambda x. x + x \) and \( \lambda x. \text{"Hello World"} \) can be distinguished by types:

\( \lambda x. x + x : \text{int} \rightarrow \text{int} \) and \( \lambda x. \text{"Hello World"} : \alpha \rightarrow \text{string} \)
Let $t$ and $u$ two $\lambda$-terms.

- Does there exist a context $C$ such that $C\langle t \rangle$ does not terminate and $C\langle u \rangle$ terminates?

- Does there exist a typing judgment $(\Gamma, L)$ such that $\Gamma \vdash t : L$ but $\Gamma \not\vdash u : L$?

This talk is about separating with Intersection Types!
Let $t$ and $u$ two $\lambda$-terms.

- Does there exist a context $C$ such that $C\langle t \rangle$ does not terminate and $C\langle u \rangle$ terminates?

- Does there exist a typing judgment $(\Gamma, L)$ such that $\Gamma \vdash t : L$ but $\Gamma \not\vdash u : L$?

This talk is about separating with **Intersection Types**!
Böhm Out Technique

Separating Terms with Contexts: let’s reduce term to their head normal form and find an appropriate context.

- $I$ and $\Omega$ are separated by the empty context $\langle \cdot \rangle$

- $x I I$ and $x \Omega I$ are separated by $(\lambda x. \langle \cdot \rangle) \pi_1$ where $\pi_1 = \lambda x. \lambda y. x$

- $y I (x \Omega I)$ and $y I (x I I)$ are separated by $(\lambda y. (\lambda x. \langle \cdot \rangle) \pi_1) \pi_2$ where $\pi_2 = \lambda x. \lambda y. y$

- $x I (x \Omega I)$ and $x I (x I I)$ are separated by ??
Böhm Out Technique

Separating Terms with Contexts: let’s reduce term to their head normal form and find an appropriate context.

- I and Ω are separated by the empty context ⟨·⟩

- xI I and xΩ I are separated by (λx.⟨·⟩)π₁ where
  \[ π₁ = λx.λy.x \]

- yI (xΩ I) and yI (x I I) are separated by (λy.(λx.⟨·⟩)π₁)π₂ where
  \[ π₂ = λx.λy.y \]

- xI (xΩ I) and xI (x I I) are separated by ??
Böhm Out Technique

Separating Terms with Contexts: let’s reduce term to their head normal form and find an appropriate context.

- $I$ and $\Omega$ are separated by the empty context $\langle \cdot \rangle$

- $xI I$ and $x\Omega I$ are separated by $(\lambda x.\langle \cdot \rangle)\pi_1$ where $\pi_1 = \lambda x.\lambda y.x$

- $yI (x\Omega I)$ and $yI (x I I)$ are separated by $(\lambda y.(\lambda x.\langle \cdot \rangle)\pi_1)\pi_2$ where $\pi_2 = \lambda x.\lambda y.y$

- $xI (x\Omega I)$ and $xI (x I I)$ are separated by $\pi_2$
Böhm Out Technique

Separating Terms with Contexts: let’s reduce term to their head normal form and find an appropriate context.

- $I$ and $\Omega$ are separated by the empty context $\langle \cdot \rangle$

- $x I I$ and $x \Omega I$ are separated by $(\lambda x.\langle \cdot \rangle)\pi_1$ where $\pi_1 = \lambda x.\lambda y.x$

- $y I (x \Omega I)$ and $y I (x I I)$ are separated by $(\lambda y.(\lambda x.\langle \cdot \rangle)\pi_1)\pi_2$ where $\pi_2 = \lambda x.\lambda y.y$

- $x I (x \Omega I)$ and $x I (x I I)$ are separated by ??
Böhm Out Technique

Separating Terms with Contexts: let’s reduce term to their head normal form and find an appropriate context.

- I and Ω are separated by the empty context \( ⟨·⟩ \)

- \( x {\underline{I}} I \) and \( x {\underline{Ω}} I \) are separated by \((\lambda x.⟨·⟩)\pi_1\) where 
  \[ \pi_1 = \lambda x.\lambda y.x \]

- \( y {\underline{I}} (x {\underline{Ω}} I) \) and \( y {\underline{I}} (x {\underline{I}} I) \) are separated by \((\lambda y.(\lambda x.⟨·⟩)\pi_1)\pi_2\) where 
  \[ \pi_2 = \lambda x.\lambda y.y \]

- \( x {\underline{I}} (x {\underline{Ω}} I) \) and \( x {\underline{I}} (x {\underline{I}} I) \) are separated by ??
Type-Böhm Out Technique

Separating Terms with Types: easier than context Böhm out to select subterms!

\[ \boxed{A \Rightarrow B, A \Rightarrow B} \]

\[ | \Gamma \vdash x \text{I} (x \text{I} I) : B \quad \text{but} \quad | \Gamma \not\vdash x \text{I} (x \Omega I) : B \]
Type-Böhm Out Technique

Separating Terms with Types: easier than context Böhm out to select subterms!

\[ x: x \Omega I \quad \text{and} \quad x: x I I \]

\[ x: [A \rightarrow B \rightarrow B, A \rightarrow B \rightarrow A] \vdash_\Delta: B \]

\[ \Gamma \vdash x: x I I : B \quad \text{but} \quad \Gamma \not\vdash x: x \Omega I : B \]
This Talk

For Weak Head Call-by-Name:

NF Bisimilarity \[\rightsquigarrow\] Derivation Transfer \[\rightarrow\] Type Equivalence
\[\leftarrow\]\ Type-Böhm Out \[\longleftarrow\]

- A *coinductive flavor* (normal form bisimulations): no approximants
- **Derivation transfer** requires non idempotent intersection types
Weak Head Multi Types

**Linear Types**
\[ L, L' ::= \star_k | M \rightarrow L \quad k \in \mathbb{N} \]

**Multi Types**
\[ M, N ::= [L_1, \ldots, L_n] \quad n \geq 0 \]

\[ \frac{x : [L] \vdash x : L}{\text{ax}} \quad \frac{\emptyset \vdash \lambda x.t : \star_k}{\lambda k} \quad \frac{\Gamma, x : M \vdash t : L}{\Gamma \vdash \lambda x.t : M \rightarrow L} \quad \lambda \]

\[ \frac{\Gamma \vdash t : [L_i]_{i \in I} \rightarrow L' \quad (\Gamma_i \vdash u : L_i)_{i \in I}}{I \text{ finite}} \quad \frac{\Gamma \uplus \Delta \vdash tu : L'}{\emptyset} \]
Type Equivalence

**Type Equivalence:** \( t \simeq_{type} u \) if \( \forall (\Gamma, L) \quad \Gamma \vdash t : L \iff \Gamma \vdash u : L \)

\( \beta \)-equivalence is included in \( \simeq_{type} \):

- by Subject Reduction and Expansion

\( \eta \)-equivalence is not included in \( \simeq_{type} \):

- \( x : [\star_0] \vdash x : \star_0 \) but \( x : [\star_0] \not\vdash \lambda y. x \ y : \star_0 \)
- \( \emptyset \vdash \lambda y. x \ y : \star_0 \) but \( \emptyset \not\vdash x : \star_0 \)
Normal form bisimilarity

The syntactical characterization is phrased in a coinductive way, using Sangiorgi's normal form bisimilarity. It coincides with the inequational theory induced by Lévy-Longo trees.

**Definition (Weak head normal form similarity)**

A relation \( R \) is a **weak head normal (whnf) form simulation** if whenever \( t R u \) holds, we have that either:

- (⊥) \( t \not\Downarrow_{wh} \), or,
- (abs) \( t \Downarrow_{wh} \lambda x.t' \) and \( u \Downarrow_{wh} \lambda x.u' \) with \( t' R u' \), or,
- (app) \( t \Downarrow_{wh} y t_1 \cdots t_k \) and 
  \( u \Downarrow_{wh} y u_1 \cdots u_k \) with \((t_i R u_i)_{i \leq k}\).

Whnf similarity, noted \( \lesssim_{nf} \), is the largest whnf simulation.
Derivation Transfer
From bisimilar terms to type equivalent types

The proof goes by induction on the size of derivations.
Assume \( t \not\sim_{\text{nf}} u \) and \( \pi \triangleright \Gamma \vdash t : L \).
By **Quantitative Subject Reduction**, \( \pi' \triangleright \Gamma \vdash n_t : L \) s.t. \( |\pi'| \leq |\pi| \).
*By case analysis on the shape of \( n_t \):*

**Application case:**
\[
\Gamma \vdash xt_1 \ldots t_{k-1} : [L_i]_{i \in I} \rightarrow L \quad (\Delta_i \vdash t_k : L_i)_{i \in I}
\]
\[
\pi' : \quad \Gamma \uplus \left( \biguplus_{i \in I} \Delta_i \right) \vdash n_t = xt_1 \cdots t_k : L
\]
\[
\leadsto \quad \sigma \triangleright \Gamma \vdash n_u : L \quad \text{as } xt_1 \cdots t_{k-1} \not\sim_{\text{nf}} xu_1 \cdots u_{k-1} \text{ and } t_k \not\sim_{\text{nf}} u_k.
\]
By **Subject Expansion**, \( \sigma' \triangleright \Gamma \vdash u : L \).
Derivation Transfer

From bisimilar terms to type equivalent types

The proof goes by induction on the size of derivations.
Assume $t \not\sim_{nf} u$ and $\pi \triangleright \Gamma \vdash t : L$.
By **Quantitative Subject Reduction**, $\pi' \triangleright \Gamma \vdash n_t : L$ s.t. $|\pi'| \leq |\pi|$.

**By case analysis on the shape of $n_t$:**

**Application case:**

$\Gamma \vdash xt_1 \cdots t_{k-1} : [L_i]_{i \in I} \rightarrow L \quad (\Delta_i \vdash t_k : L_i)_{i \in I}$

$\pi' : \quad \Gamma \cup (\bigcup_{i \in I} \Delta_i) \vdash n_t = xt_1 \cdots t_k : L$

$\leadsto \quad \sigma \triangleright \Gamma \vdash n_u : L \quad$ as $xt_1 \cdots t_{k-1} \not\sim_{nf} xu_1 \cdots u_{k-1}$ and $t_k \not\sim_{nf} u_k$.

By Subject Expansion, $\sigma' \triangleright \Gamma \vdash u : L$. 
Type Equivalence & Compositionality

Type equivalence is **compositional**.

- $t \lessdot_{\text{type}} u$ and $s \lessdot_{\text{type}} r$ implies $ts \lessdot_{\text{type}} ur$
- $t \lessdot_{\text{type}} u$ implies $\lambda x.t \lessdot_{\text{type}} \lambda x.u$

**Proposition (Soundness wrto Contextual Equivalence)**

*If $t \lessdot_{\text{type}} u$ then $t \leq^{wh}_{C} u$*

Side note: $t \leq^{wh}_{C} u$ does not imply $t \lessdot_{\text{type}} u$ (problems with $\eta$...)
Type equivalence is **compositional**.

- $t \simeq_{\text{type}} u$ and $s \simeq_{\text{type}} r$ implies $ts \simeq_{\text{type}} ur$
- $t \simeq_{\text{type}} u$ implies $\lambda x.t \simeq_{\text{type}} \lambda x.u$

**Proposition (Soundness wrto Contextual Equivalence)**

If $t \simeq_{\text{type}} u$ then $t \leq_{\text{wh}} u$

Side note: $t \leq_{\text{wh}} u$ does not imply $t \simeq_{\text{type}} u$ (problems with $\eta$...)
NF Bisimilarity \iff \text{Type-Böhm Out} \iff \text{Type Equivalence}

\textbf{Coinduction principle:} \bowtie_{\text{type}} \text{ is a whnf simulation} \Rightarrow \bowtie_{\text{type}} \subseteq \bowtie_{\text{nf}}

We show that \bowtie_{\text{type}} \text{ is a whnf simulation by contrapositive:}
Let \( t \) and \( u \) such that \( t \downarrow_{\text{wh}} n_t \) and \( u \downarrow_{\text{wh}} n_u \) with different normal forms

\( \rightsquigarrow \text{Build a Separating Type} \)
Separating Terms through Multi Types

Some examples

1. **Separating any abstraction** \( \lambda x.t \) **and any applied** \( nf \times t_1 \cdots t_k \):

\[
\Gamma \vdash \lambda x.t : *_0 \quad \lambda_0 \quad \vdash \Gamma \vdash x t_1 \cdots t_k : *_0
\]

Then \( \Gamma \) must have the shape:
\[
\Gamma = \emptyset \quad \text{or} \quad \Gamma = x : [M_1 \rightarrow \cdots , \cdots ]
\]

2. **Separating a variable** \( x \) **and an applied** \( nf \times t \):

3. **Separating different abstractions**:

4. **Separating applied** \( nf \) **where arguments differ, say** \( xt \) **and** \( xu \):
Separating Terms through Multi Types

Some examples

1. *Separating any abstraction* $\lambda x.t$ *and any applied nf* $x \, t_1 \cdots t_k$:

2. *Separating a variable* $x$ *and an applied nf* $xt$:

   \[
   \Gamma \vdash x : \star_0\quad \vdash \quad \vdash \quad \vdash \quad \vdash \\
   \Gamma \supset x : [\star_0]
   \]

   Then $\Gamma$ must resemble:

   \[
   \Gamma = x : [\star_0]\quad \quad \Gamma \supset x : [M \to \cdots , \cdots ]
   \]

3. *Separating different abstractions*:

4. *Separating applied nf where arguments differ, say* $xt$ *and xu*:

   \[
   \]
Separating Terms through Multi Types

Some examples

1. *Separating any abstraction* $\lambda x.t$ *and any applied nf* $x \cdot t_1 \cdots t_k$:

2. *Separating a variable* $x$ *and an applied nf* $xt$:

3. *Separating different abstractions*:

   Suppose we have $\Gamma, x : M \vdash t : L$ but $\Gamma, x : M \not\vdash u : L$

   
   \[\Gamma, x : M \vdash t : L\]

   \[\Gamma \vdash \lambda x. t : M \rightarrow L\]

   \[\lambda \quad \text{If} \quad \Gamma \vdash \lambda x. u : M \rightarrow L\]

   Then it must start with:

   \[\Gamma, x : M \vdash u : L\]

   \[\Gamma \vdash \lambda x. u : M \rightarrow L\]

4. *Separating applied nf where arguments differ, say* $xt$ *and* $xu$:
Separating Terms through Multi Types

Some examples

1. *Separating any abstraction* $\lambda x. t$ *and any applied* $nf x t_1 \cdots t_k$:
2. *Separating a variable* $x$ *and an applied* $nf xt$:
3. *Separating different abstractions*:
4. *Separating applied* $nf$ *where arguments differ, say* $xt$ *and* $xu$:

Suppose we have $\Gamma \vdash t : L$ *but* $\Gamma \not\vdash u : L$

\[
\begin{array}{c}
\Delta \vdash x : [L] \leadsto_k \Gamma \vdash t : L \\
\hline
\Gamma \uplus \Delta \vdash x t : \star_i
\end{array}
\]

Then it must start with:

\[
\begin{array}{c}
\Delta' \vdash x : [L_i]_i \leadsto_k (\Gamma_i \vdash u : L_i)_i \\
\hline
\Gamma \uplus \Delta' \vdash x u : \star_i
\end{array}
\]

If $k$ is well chosen, then $\Delta' = x : ([L] \leadsto \star_k]$
Factorizing the need of countable atoms

*Compositionality* of $\mathcal{L}_{\text{type}}$ is the only part that requires many distinguishable ground types!

Intuitively: $t \not\equiv_{\text{type}} u$ or $s \not\equiv_{\text{type}} r$ implies $ts \not\equiv_{\text{type}} ur$

- **Left difference**: $x t_1 \cdots t_k \not\equiv_{\text{type}} y u_1 \cdots u_{k'}$ implies $x t_1 \cdots t_k s \not\equiv_{\text{type}} y u_1 \cdots u_{k'} r$.
- **Right difference**: $x t_1 \cdots t_k \not\equiv_{\text{type}} y u_1 \cdots u_{k'}$ and $s \not\equiv_{\text{type}} r$ implies $x t_1 \cdots t_k s \not\equiv_{\text{type}} y u_1 \cdots u_{k} r$. 
Factorizing the need of countable atoms

Compositionality of $\mathcal{L}_\text{type}$ is the only part that requires many distinguishable ground types!

Intuitively: $t \mathcal{L}_\text{type} u$ or $s \mathcal{L}_\text{type} r$ implies $ts \mathcal{L}_\text{type} ur$

- **Left difference:** $\times t_1 \cdots t_k \mathcal{L}_\text{type} y u_1 \cdots u_{k'}$ implies $\times t_1 \cdots t_k s \mathcal{L}_\text{type} y u_1 \cdots u_{k'} r$.

- **Right difference:** $\times t_1 \cdots t_k \mathcal{L}_\text{type} y u_1 \cdots u_{k'}$ and $s \mathcal{L}_\text{type} r$ implies $\times t_1 \cdots t_k s \mathcal{L}_\text{type} y u_1 \cdots u_k r$. 
Conclusion

NF Bisimilarity  Derivation Transfer  Type Equivalence
                   Type-Böhm Out

Perspectives:
▶ Towards Call-by-Value equivalences, where NF bisimilarities are more involved
▶ Can we adapt type equivalence to reach full abstraction?

            Compositionality?

▶ Can Type-Böhm out help to simplify (Context-)Böhm out?

Thank you!
Conclusion

NF Bisimilarity $\xrightarrow{\text{Derivation Transfer}}$ Type Equivalence
$\xleftarrow{\text{Type-Böhm Out}}$

Perspectives:

- Towards Call-by-Value equivalences, where NF bisimilarities are more involved
- Can we adapt type equivalence to reach full abstraction?

Ctx. Equiv. $\xrightarrow{\text{Definability?}}$ Inhabited Type Equivalence
$\xleftarrow{\text{Compositionality?}}$

- Can Type-Böhm out help to simplify (Context-)Böhm out?

Thank you!