

Normal Form Bisimulations by Value

Beniamino Accattoli¹, Adrienne Lancelot^{1,2} & Claudia Faggian²

¹Inria & LIX, École Polytechnique
²IRIF, Université Paris Cité & CNRS

November 21st 2023 - University of Birmingham

Outline

Programming Languages & Program Equivalence

Equivalence of Programs

Plotkin's Style Call-by-Value

Lassen's Call-by-Value Bisimilarity

An Open Call-by-Value Calculus: The Value Substitution Calculus

(New) Normal Form Bisimilarities

Type Equivalence

Conclusion and Discussion

Programming Languages

Program equivalence in the λ -calculus

We are interested in studying **program equivalence** for **functional programming languages**.

Via the **lambda-calculus**, seen as a **mathematical model** of programming languages.

What makes two lambda terms **equivalent**?

Programming Languages

Two main paradigms

There are **various** notions of program equivalence which depends on the various **dialects** of the λ -calculus.

And even more variants if we were to consider effects, or others additions to the calculus.

- ▶ *Call-by-Name* is the variant most used in theoretical studies.
- ▶ *Call-by-Value* is a more accurate model of **functional** programming languages.

Function **arguments are evaluated first**.

Programming Languages

Open terms

Programs are usually defined as **closed** terms.

A term is closed if it has no free variables.

Closed terms are **expressive enough** to model all computable functions.

But to study certain subjects, such as the implementation model of Coq, one needs open terms.

Programming Languages

Call-by-Value theory

Call-by-Value was formalized by Plotkin [Plot75].

Its theory is well-behaved for closed terms, but is **not very satisfactory on open terms**.

In the literature, there are some propositions to **enhance the open Call-by-Value** setting – usually extensions of Plotkin's CbV:

- ▶ *Moggi's work* on computational lambda-calculus (with lets) [Mog89].
- ▶ *Open Call-by-Value*. A recent advance towards a generalized theory, related with Linear Logic [AG16].

An operational characterization for meaningless and meaningful terms?

Programming Languages

Meaningless and meaningful terms

In lambda-calculus, not all divergent terms diverge in the same way.

- ▶ *Meaningful*. Core example : Fix-points operators, Y and Θ
Some terms may be divergent but still **meaningful**, by producing increasing information.
- ▶ *Meaningless*. All terms equivalent to $\delta\delta$ are **meaningless**.
Convolute definition...

In Call-by-Name, meaningful = solvable (head normalizable).

In Plotkin's Call-by-Value, meaningful \neq ??-normalizable:

$\Omega_L = (\lambda x.\delta)(yy)\delta$ is a **meaningless normal form**.

Outline

Programming Languages & Program Equivalence

Equivalence of Programs

Plotkin's Style Call-by-Value

Lassen's Call-by-Value Bisimilarity

An Open Call-by-Value Calculus: The Value Substitution Calculus

(New) Normal Form Bisimilarities

Type Equivalence

Conclusion and Discussion

Equivalence of Programs

Contextual Equivalence

Two terms that **behave** the same **in any given environment**, are *contextually equivalent*.

- ▶ A **natural** notion.
- ▶ In practice, **not usable**.
Unable to prove contextual equivalence for the fixed point combinators.
- ▶ Which depends on the definition of **dialect**.
Call-by-Name and Call-by-Value have different notions of contextual equivalence.

Equivalence of Programs

Generalities

What are equivalent programs ?

Three important properties for a relation \mathcal{R} on terms:

Equivalence	Reflexivity	Symmetry	Transitivity
Compatibility	$t \mathcal{R} u$	\Rightarrow	$C\langle t \rangle \mathcal{R} C\langle u \rangle$
Adequacy	$t \mathcal{R} u$	\Rightarrow	$t \Downarrow$ iff $u \Downarrow$
Conversion	$t =_{\beta} u$	\Rightarrow	$t \mathcal{R} u$

If a relation is **compatible** and **adequate**, then it is **included in contextual equivalence**.

If an **equivalence** relation is **compatible** and includes **conversion**, then it is **an equational theory**.

Equivalence of Programs

Normal Form Bisimilarity

Normal form bisimilarity [San94] can be seen as a **technique to prove contextual equivalence**.

Normal form bisimilarity states program equivalence for λ -terms by looking at the **structure of their normal forms**.

As an example, in Call-by-Value, we relate $\lambda x.t$ and $\lambda x.t'$ by relating t and t'

This is also called *open bisimilarity* because we need to deal with open terms.

Which is inherent when inspecting the body of functions, that is, moving from an closed term $\lambda x.t$ to an **open** term t .

Equivalence of Programs

Normal Form Bisimilarity

normal form bisimilarity \subseteq *contextual equivalence*

- ▶ Similarly written programs behave the same in any environment.
- ▶ The converse is not obvious and will depend on how normal form bisimilarities inspect normal forms.

Normal Form Bisimulations by Name

Standard normal form bisimulations

In Call-by-Name, normal form bisimulations have been introduced by Sangiorgi [San94], coming from Pi-calculus bisimulations.

- ▶ Refined by Lassen [Las99] and related with **Böhm and Lévy-Longo trees**
- ▶ **Identify meaningless** because they use (weak) head reduction
- ▶ Adding η -equivalence, yields a **fully abstract** program equivalence. (Nakajima trees)

Normal Form Bisimulations by Value

State-of-the-art normal form bisimulations by value

In the literature, a **Call-by-Value normal form bisimilarity**¹ has been developed by Lassen [Las05], based on Plotkin's CbV calculus.

Eager Normal Form Bisimilarity \simeq_{enf}

- ▶ Validates Moggi's laws ($\mathbb{I}t \equiv_{lid} t$ for all t)
 $\mathbb{I}(yy) \simeq_{enf} yy, \dots$
- ▶ Differentiates between different **meaningless** terms
 $\Omega_L \not\simeq_{enf} \Omega$

The second point is the starting point of our work: to create a normal form bisimilarity that identifies meaningless terms.

Going back to Pi-calculus, it is possible to characterize Lassen's Enf in Pi [DHS22].

¹But this nf bisimilarity is not defined as CbN bisimilarities are.

Normal Form Bisimulations by Value

Four program equivalences

Overview: How to **adapt** normal form bisimulations to Call-by-Value ?

- ▶ (Natural) **Naive** CbV Normal Form Bisimilarity
- ▶ (State-of-the-art) Lassen's Eager Normal Form Bisimilarity
- ▶ (New) **Net Bisimilarity**
- ▶ (Goal) Relational Semantics: Type Equivalence

Contributions

Naive normal form bisimilarity

By rephrasing Call-by-Name weak head normal form bisimulations (that is Sangiorgi's open bisimulation or Lévy-Longo bisimulation) in Call-by-Value, we get:

Naive Call-by-Value Normal Form Bisimulation \simeq_{nai}

- ▶ Usable for some infinitary normal forms
Curry's and Turing's fix-points combinators are naive CbV normal form bisimilar
- ▶ Not much more...
 $I(yy) \not\approx_{nai} yy$, $\Omega_L \not\approx_{nai} \Omega$, $I(I(yy)) \not\approx_{nai} I(yy)$, ...

Contributions

Net Bisimilarity

We developed a **new** CbV normal form bisimilarity, relying on the theory of **Open Call-by-Value**.

More precisely, the Value Substitution Calculus [AP12].

Net Bisimilarity \simeq_{net}

- ▶ By construction, it **identifies all meaningless terms**.

$$\Omega_L \simeq_{net} \Omega$$

- ▶ It does not subsume Lassen's enf bisimilarity.

$$I(yy) \not\sim_{net} (yy)$$

Contributions

Technical Proof

Soundness wrto Contextual Equivalence

A crucial point is to **prove compatibility**.

Compatibility $t \mathcal{R} u \Rightarrow C\langle t \rangle \mathcal{R} C\langle u \rangle$

► **Lassen's method:**

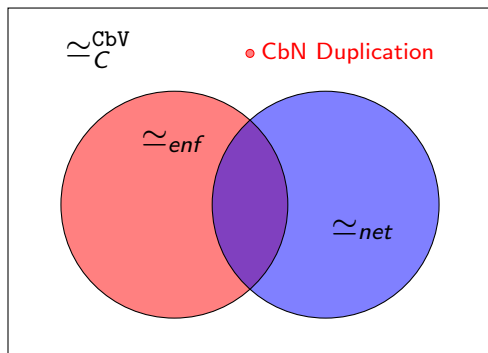
A simpler Howe's method, used in another paper [Las99] by Lassen about call-by-name normal form bisimilarities.

Introduce a contextual closure, then **prove the contextual closure of a bisimulation is a bisimulation!** By coinduction, the contextual closure of the bisimilarity coincides with the bisimilarity.

Normal Form Bisimulations by Value

Soundness and Incompleteness wrto Contextual Equivalence

Both \simeq_{enf} and \simeq_{net} are included **strictly** in contextual equivalence.

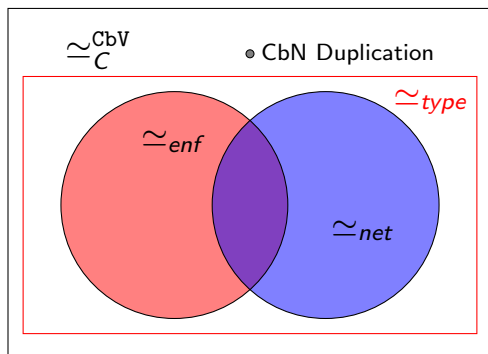


$$\text{(CbN duplication)} \quad \delta(yy) \simeq_C^{CbV} (yy)(yy)$$

Normal Form Bisimulations by Value

Soundness and Incompleteness wrto Relational Semantics

Both bisimilarities are also included in the equational theory induced by Ehrhard's Call-by-Value relational semantics. Types here refer to *intersection types*, which are a syntactic presentation of the denotational –relational– semantics.



Type Equivalence \approx_{type}

Contribution

Results

The two bisimilarities are **orthogonal**.

- ▶ Lassen's enf bisimilarity validates the identity rule.

$$I(t) \simeq_{enf} t \text{ for any term } t$$

- ▶ Net bisimilarity identifies meaningless terms.

$$\Omega_L \simeq_{net} \Omega$$

Moggi's laws or theory of Open Call-By-Value, but not both.

Mixing both could mean **matching type equivalence!**

Outline

Programming Languages & Program Equivalence

Equivalence of Programs

Plotkin's Style Call-by-Value

Lassen's Call-by-Value Bisimilarity

An Open Call-by-Value Calculus: The Value Substitution Calculus

(New) Normal Form Bisimilarities

Type Equivalence

Conclusion and Discussion

General Notions

Call-by-Value

We refer to the following calculus as *Plotkin's Call-by-Value*.

$$\begin{array}{ll} \text{TERMS} & t, u ::= v \mid tu \\ \text{VALUES} & v, v' ::= x \mid \lambda x. t \end{array}$$

The CbV reduction restricts β -redexes to abstractions applied to values.

$$\text{WEAK CONTEXTS } E ::= \langle \cdot \rangle \mid Et \mid tE$$

Weak reduction \rightarrow_w is defined by **Weak** contextual closure of the top-level rule \mapsto_{β_v} .

$$\begin{array}{ll} \beta_v\text{-REDUCTION} & \text{CONTEXTUAL CLOSURE} \\ (\lambda x. t)v \mapsto_{\beta_v} t\{x \leftarrow v\} & E\langle t \rangle \rightarrow_w E\langle t' \rangle \quad \text{if } t \mapsto_{\beta_v} t' \end{array}$$

General Notions

Contextual Equivalence

For a reduction \rightarrow , we define **big-step evaluation** \Downarrow by:

- ▶ if $t \rightarrow^k n$ and n is a normal form, then $t \Downarrow^k n$.
- ▶ if t diverges, $t \not\Downarrow$.

Definition (Contextual Equivalence)

We define contextual equivalence \simeq_C as follows:

$$t \simeq_C t' \text{ if for all } C \text{ closing}^2 \text{ contexts of } t \text{ and } t', \\ C\langle t \rangle \Downarrow \iff C\langle t' \rangle \Downarrow.$$

More precisely, we consider \simeq_C^{CbV} where the evaluation is \Downarrow_{CbV} associated with the weak reduction \rightarrow_w .

² $C\langle t \rangle$ and $C\langle t' \rangle$ are closed terms

Normal Form Bisimulations by Name

Weak head normal form bisimulations

A relation \mathcal{R} is a **weak head normal form bisimulation** if, whenever $t \mathcal{R} t'$ then one of the following cases hold:

(wh 1) t and t' have no \rightarrow_{wh} -normal forms.

(wh 2) $t \rightarrow_{wh}^* \lambda x. t_1$ and $t' \rightarrow_{wh}^* \lambda x. t'_1$ with $t_1 \mathcal{R} t'_1$

(wh 3) $t \rightarrow_{wh}^* x t_1 \dots t_k$ and $t' \rightarrow_{wh}^* x t'_1 \dots t'_k$ with $(t_i \mathcal{R} t'_i)_{i \leq k}$

(wh 3.1) $t \rightarrow_{wh}^* x$ and $t' \rightarrow_{wh}^* x$

(wh 3.2) $t \rightarrow_{wh}^* n u$ and $t' \rightarrow_{wh}^* n' u'$

with $n \mathcal{R} n'$ and $u \mathcal{R} u'$

Normal Form Bisimulations by Name

Weak head normal form bisimulations

A relation \mathcal{R} is a **weak head normal form bisimulation** if, whenever $t \mathcal{R} t'$ then one of the following cases hold:

(wh 1) t and t' have no \rightarrow_{wh} -normal forms.

(wh 2) $t \rightarrow_{wh}^* \lambda x. t_1$ and $t' \rightarrow_{wh}^* \lambda x. t'_1$ with $t_1 \mathcal{R} t'_1$

(wh 3) $t \rightarrow_{wh}^* x t_1 \dots t_k$ and $t' \rightarrow_{wh}^* x t'_1 \dots t'_k$ with $(t_i \mathcal{R} t'_i)_{i \leq k}$

(wh 3.1) $t \rightarrow_{wh}^* x$ and $t' \rightarrow_{wh}^* x$

(wh 3.2) $t \rightarrow_{wh}^* n u$ and $t' \rightarrow_{wh}^* n' u'$

with $n \mathcal{R} n'$ and $u \mathcal{R} u'$

Naive CbV normal form bisimilarity

A relation \mathcal{R} is a **naive Call-by-Value normal form bisimulation** if, whenever $t \mathcal{R} t'$ then one of the following cases hold:

$$\text{(nai 1)} \quad t \Downarrow_w \quad \text{and} \quad t' \Downarrow_w$$

$$\text{(nai 2)} \quad t \Downarrow_w \times \quad \text{and} \quad t' \Downarrow_w \times$$

$$\text{(nai 3)} \quad t \Downarrow_w \lambda x. t_1 \quad \text{and} \quad t' \Downarrow_w \lambda x. t'_1 \\ \text{with } t_1 \mathcal{R} t'_1$$

$$\text{(nai 4)} \quad t \Downarrow_w n_1 n_2 \quad \text{and} \quad t' \Downarrow_w n'_1 n'_2 \\ \text{with } n_1 \mathcal{R} n'_1 \text{ and } n_2 \mathcal{R} n'_2$$

Naive CbV normal form bisimilarity is defined by co-induction, as the largest net bisimulation.

Naive CbV Normal Form Bisimilarity

What for?

- ▶ Accounts for infinitary behavior.
The fix-points operators are naively bisimilar.
- ▶ Does not fit in any improvement of Call-by-Value for open terms
 - ▶ (Moggi) $I(yy) \not\sim_{nai} yy$
 - ▶ (Meaningless) $\Omega_L \not\sim_{nai} \Omega$

Outline

Programming Languages & Program Equivalence

Equivalence of Programs

Plotkin's Style Call-by-Value

Lassen's Call-by-Value Bisimilarity

An Open Call-by-Value Calculus: The Value Substitution Calculus

(New) Normal Form Bisimilarities

Type Equivalence

Conclusion and Discussion

Lassen's Enf Bisimilarity

Left Weak Call-by-Value reduction

Lassen uses a **restriction** on Plotkin's Call-by-Value Weak reduction, which we call \rightarrow_{1as} .

$(xx)(\delta\delta)$ is a normal form for \rightarrow_{1as} but $(xx)(\delta\delta) \rightarrow_w (xx)(\delta\delta)$

Lassen differentiates between **meaningless** terms.

Left Normal Forms

We define precisely this reduction **Lassen's Left** reduction, noted \rightarrow_{las} .

LEFT CONTEXTS $L ::= \langle \cdot \rangle \mid vL \mid Lt$

RULE AT TOP LEVEL LEFT CONTEXTUAL CLOSURE
 $(\lambda x.t)v \mapsto_{\beta_v} t\{x \leftarrow v\}$ $L\langle t \rangle \rightarrow_{\text{las}} L\langle u \rangle$ if $t \mapsto_{\beta_v} u$

Lassen defines a **left normal forms grammar**.

Lemma (Lassen)

Terms are either values or admit a unique decomposition $L\langle vv \rangle$.

LEFT NORMAL FORMS $n ::= v \mid L\langle xv \rangle$

Lassen's Enf Bisimilarity

Eager normal form simulation

A relation \mathcal{R} between λ -terms is an **eager normal form (enf) bisimulation** [Las05] if, whenever $t \mathcal{R} t'$ then one of the following clauses holds:

$$\text{(enf 1)} \quad t \Downarrow_{\text{las}} \quad \text{and} \quad t' \Downarrow_{\text{las}}$$

$$\text{(enf 2)} \quad t \Downarrow_{\text{las}} \times \quad \text{and} \quad t' \Downarrow_{\text{las}} \times$$

$$\text{(enf 3)} \quad t \Downarrow_{\text{las}} \lambda x. t_1 \quad \text{and} \quad t' \Downarrow_{\text{las}} \lambda x. t'_1 \\ \text{with } t_1 \mathcal{R} t'_1$$

$$\text{(enf 4)} \quad t \Downarrow_{\text{las}} L\langle \mathbf{xv} \rangle \quad \text{and} \quad t' \Downarrow_{\text{las}} L'\langle \mathbf{xv}' \rangle \\ \text{with } \mathbf{v} \mathcal{R} \mathbf{v}' \text{ and } L\langle \mathbf{z} \rangle \mathcal{R} L'\langle \mathbf{z} \rangle \\ \text{where } \mathbf{z} \text{ is not free in } L \text{ or } L'$$

Enf bisimilarity, noted \simeq_{enf} , is defined by co-induction as the largest enf bisimulation.

Lassen's Enf Bisimilarity

Enf bisimilarity

We say that t and t' are **enf bisimilar** whenever $t \simeq_{enf} t'$.

Theorem

- ▶ *Enf bisimilarity is compatible*
- ▶ $\forall t, t', t \simeq_{enf} t' \Rightarrow t \simeq_C t'$

As an extension of \simeq_{nai} , enf bisimilarity accounts for infinitary behavior.

Turing's and Curry's fix-points combinators are enf bisimilar.

Examples of enf bisimilar terms

Moggi's laws

The following **equations of Moggi's** untyped computational λ -calculus (without lets) are satisfied by enf bisimilarity:

- ▶ (id) $(\lambda x.x)t \equiv_{lid} t$
- ▶ (assoc) $(\lambda x.t)((\lambda y.u)s) \equiv_{ass} (\lambda y.(\lambda x.t)u)s$ if $y \notin \text{fv}(u)$
- ▶ (let.1) $tu \equiv_{lad} (\lambda x.xu)t$ if $x \notin \text{fv}(u)$
- ▶ (let.2) $vt \equiv_{rad} (\lambda x.vx)t$ if $x \notin \text{fv}(v)$.

Shortcomings of enf bisimilarity

Enf bisimilarity is not complete with respect to contextual equivalence (\simeq_C^{CbV}).

▶ (*extensionality*) $y \not\sim_{enf} \lambda x. yx$

This is fixed by Lassen by adding η_v equivalence to enf.

▶ (*meaningless left*³) $(xx)(\delta\delta) \not\sim_{enf} \delta\delta$

▶ (*meaningless*) $(\lambda x. \delta)(yy)\delta \not\sim_{enf} \delta\delta$

³Here this meaningless term is normal because of Lassen's choice to reduce left to right, not because Call-by-Value evaluation is stuck

Outline

Programming Languages & Program Equivalence

Equivalence of Programs

Plotkin's Style Call-by-Value

Lassen's Call-by-Value Bisimilarity

An Open Call-by-Value Calculus: The Value Substitution Calculus

(New) Normal Form Bisimilarities

Type Equivalence

Conclusion and Discussion

Value Substitution Calculus

An Open Call-by-Value Viewpoint

The **Value Substitution Calculus** (VSC) was introduced by Accattoli and Paolini [AP12].

It is a presentation of **Open Call-by-Value** [AG16].

It has a **well-behaved rewriting theory** and an ability to **circumvent stuck redexes**.

Value Substitution Calculus

A calculus with explicit substitutions

$$\begin{array}{l} \text{TERMS} \quad t, u, s \quad ::= \quad v \mid tu \mid t[x \leftarrow u] \\ \text{VALUES} \quad \quad v \quad ::= \quad x \mid \lambda x. t \end{array}$$

Explicit substitutions are sometimes written as `let $x = u$ in t` .

We consider Weak VSC, which is given using contextual closure over the **evaluation contexts** – weak contexts extended for the new construct of explicit substitutions.

$$E \quad ::= \quad \langle \cdot \rangle \mid tE \mid Et \mid E[x \leftarrow u] \mid t[x \leftarrow E]$$

Value Substitution Calculus

Reduction in two steps

There are **two** reduction rules: the **multiplicative** \rightarrow_m rule and the **exponential** \rightarrow_e rule.

SIMPLIFIED RULES

$$(\lambda x.t)u \rightarrow_m t[x \leftarrow u]$$

$$t[x \leftarrow v] \rightarrow_e t\{x \leftarrow v\}$$

The β_v -reduction is fractioned: $ll \rightarrow_{\beta_v} l$ but $ll \rightarrow_m x[x \leftarrow l] \rightarrow_e l$

Value Substitution Calculus

Reduction at a distance

Substitution Contexts are lists of substitutions:

$$S ::= \langle \cdot \rangle \mid S[x \leftarrow u]$$

Actual reduction rules are **at a distance**.

REDUCTION RULES

$$\begin{aligned} S\langle \lambda x. t \rangle u &\rightarrow_m S\langle t[x \leftarrow u] \rangle \\ t[x \leftarrow S\langle v \rangle] &\rightarrow_e S\langle t\{x \leftarrow v\} \rangle \end{aligned}$$

We refer to Weak VSC as VSC, and note the reduction \rightarrow_{vsc} .

Value Substitution Calculus

Reduction at a distance

Distance = Needed Permutations to unstuck redexes

$$(\lambda x_1. (\lambda x. t))(y u) v \rightarrow_m (\lambda x. t)[x_1 \leftarrow y u] v \rightarrow_m t[x \leftarrow v][x_1 \leftarrow y u] \rightarrow_e \dots$$

$$\begin{aligned} (\lambda x. t)((\lambda x_2. v)(z u)) &\rightarrow_m (\lambda x. t)(v[x_2 \leftarrow z u]) \rightarrow_m t[x \leftarrow v[x_2 \leftarrow z u]] \\ &\rightarrow_e t\{x \leftarrow v\}[x_2 \leftarrow z u] \rightarrow_{vsc} \dots \end{aligned}$$

Value Substitution Calculus

VSC has a powerful reduction

$$\begin{aligned}\Omega_L &= (\lambda x. \delta)(yy)\delta \rightarrow_m \delta[x \leftarrow yy]\delta \rightarrow_m \delta\delta[x \leftarrow yy] \\ &\rightarrow_e \delta\delta[x \leftarrow yy] \rightarrow_m \rightarrow_e \delta\delta[x \leftarrow yy] \rightarrow_m \rightarrow_e \dots\end{aligned}$$

$$\begin{aligned}\Omega_R &= \delta((\lambda x. \delta)(yy)) \rightarrow_m \delta(\delta[x \leftarrow yy]) \rightarrow_m \delta\delta[x \leftarrow yy] \\ &\rightarrow_e \delta\delta[x \leftarrow yy] \rightarrow_m \rightarrow_e \delta\delta[x \leftarrow yy] \rightarrow_m \rightarrow_e \dots\end{aligned}$$

Theorem (Operational Characterization of Meaninglessness [AP12])

A term t is meaningless iff $t \Downarrow_{\text{VSC}}$.

Outline

Programming Languages & Program Equivalence

Equivalence of Programs

Plotkin's Style Call-by-Value

Lassen's Call-by-Value Bisimilarity

An Open Call-by-Value Calculus: The Value Substitution Calculus

(New) Normal Form Bisimilarities

Type Equivalence

Conclusion and Discussion

Naive CbV-VSC normal form bisimilarity

A relation \mathcal{R} is a naive CbV-VSC normal form bisimulation if, whenever $t \mathcal{R} t'$ then one of the following cases hold:

$$\text{(nai 1)} \quad t \Downarrow_{\text{VSC}} \quad \text{and} \quad t' \Downarrow_{\text{VSC}}$$

$$\text{(nai 2)} \quad t \Downarrow_{\text{VSC}} \times \quad \text{and} \quad t' \Downarrow_{\text{VSC}} \times$$

$$\text{(nai 3)} \quad t \Downarrow_{\text{VSC}} \lambda x. t_1 \quad \text{and} \quad t' \Downarrow_{\text{VSC}} \lambda x. t'_1 \\ \text{with } t_1 \mathcal{R} t'_1$$

$$\text{(nai 4)} \quad t \Downarrow_{\text{VSC}} n_1 n_2 \quad \text{and} \quad t' \Downarrow_{\text{VSC}} n'_1 n'_2 \\ \text{with } n_1 \mathcal{R} n'_1 \text{ and } n_2 \mathcal{R} n'_2$$

$$\text{(nai 5)} \quad t \Downarrow_{\text{VSC}} n_1[x \leftarrow n_2] \quad \text{and} \quad t' \Downarrow_{\text{VSC}} n'_1[x \leftarrow n'_2] \\ \text{with } n_1 \mathcal{R} n'_1 \text{ and } n_2 \mathcal{R} n'_2$$

Naive CbV-VSC normal form bisimilarity is defined by co-induction, as the largest naive CbV-VSC bisimulation.

Naive CbV-VSC normal form bisimilarity

Examples and shortcoming

- ▶ All meaningless terms are naive CbV-VSC bisimilar.
- ▶ However, this naive attempt is still too rigid:
 - ▶ Does not validate Moggi's identity rule
$$x[x \leftarrow t] \not\sim_{nai} t$$
 - ▶ And does not allow permutation between applications and explicit substitutions
$$xI[x \leftarrow yI] \not\sim_{nai} x[x \leftarrow yI]I$$

Value Substitution Calculus & Proof Nets

Structural Equivalence

The VSC was introduced to study the relationship between CbV and [Linear Logic](#).

From this correspondance yields a program equivalence:

(Structural Equivalence) $t \equiv_{str} u$ if $\text{ProofNet}(t) = \text{ProofNet}(u)$

Structural Equivalence is equivalent to a [syntactic axiomatization](#):

$$\begin{array}{lll} (ts)[x \leftarrow u] & \equiv_{\sigma_1} & t[x \leftarrow u]s & \text{if } x \notin \text{fv}(s) \\ (ts)[x \leftarrow u] & \equiv_{ex\sigma_3} & ts[x \leftarrow u] & \text{if } x \notin \text{fv}(t) \\ t[x \leftarrow u][y \leftarrow s] & \equiv_{ass} & t[x \leftarrow u[y \leftarrow s]] & \text{if } y \notin \text{fv}(t) \\ t[y \leftarrow s][x \leftarrow u] & \equiv_{com} & t[x \leftarrow u][y \leftarrow s] & \text{if } x \notin \text{fv}(s) \text{ and } y \notin \text{fv}(u) \end{array}$$

4

⁴ \equiv_{str} is the smallest equivalence relation, which includes these equalities and that is compatible.

Value Substitution Calculus & Proof Nets

Structural Equivalence

The VSC was introduced to study the relationship between CbV and [Linear Logic](#).

From this correspondance yields a program equivalence:

(Structural Equivalence) $t \equiv_{str} u$ if $\text{ProofNet}(t) = \text{ProofNet}(u)$

Structural Equivalence is equivalent to a [syntactic axiomatization](#):

$$\begin{array}{lll} (ts)[x \leftarrow u] & \equiv_{\sigma_1} & t[x \leftarrow u]s & \text{if } x \notin \text{fv}(s) \\ (ts)[x \leftarrow u] & \equiv_{ex\sigma_3} & ts[x \leftarrow u] & \text{if } x \notin \text{fv}(t) \\ t[x \leftarrow u][y \leftarrow s] & \equiv_{ass} & t[x \leftarrow u][y \leftarrow s] & \text{if } y \notin \text{fv}(t) \\ t[y \leftarrow s][x \leftarrow u] & \equiv_{com} & t[x \leftarrow u][y \leftarrow s] & \text{if } x \notin \text{fv}(s) \text{ and } y \notin \text{fv}(u) \end{array}$$

4

⁴ \equiv_{str} is the smallest equivalence relation, which includes these equalities and that is compatible.

Net bisimilarity

A relation \mathcal{R} is a **net bisimulation** if, whenever $t \mathcal{R} t'$ then one of the following cases hold:

$$\text{(nai 1)} \quad t \not\Downarrow_{\text{vsc}} \quad \text{and} \quad t' \not\Downarrow_{\text{vsc}}$$

$$\text{(nai 2)} \quad t \Downarrow_{\text{vsc}} x \quad \text{and} \quad t' \Downarrow_{\text{vsc}} x$$

$$\text{(nai 3)} \quad t \Downarrow_{\text{vsc}} \lambda x. t_1 \quad \text{and} \quad t' \Downarrow_{\text{vsc}} \lambda x. t'_1 \\ \text{with } t_1 \mathcal{R} t'_1$$

$$\text{(nai 4)} \quad t \Downarrow_{\text{vsc}} n_1 n_2 \quad \text{and} \quad t' \Downarrow_{\text{vsc}} n' \equiv_{\text{str}} n'_1 n'_2 \\ \text{with } n_1 \mathcal{R} n'_1 \text{ and } n_2 \mathcal{R} n'_2$$

$$\text{(nai 5)} \quad t \Downarrow_{\text{vsc}} n_1 [x \leftarrow n_2] \quad \text{and} \quad t' \Downarrow_{\text{vsc}} n' \equiv_{\text{str}} n'_1 [x \leftarrow n'_2] \\ \text{with } n_1 \mathcal{R} n'_1 \text{ and } n_2 \mathcal{R} n'_2$$

Net bisimilarity is defined by co-induction, as the largest net bisimulation.

Net Bisimilarity is Compatible

Proof

Lemma

Structural equivalence \equiv_{str} verifies:

1. *Strong commutation: if $t \equiv_{str} u$ and $t \rightarrow_{vsc} t'$ then $u \rightarrow_{vsc} u'$ and $t' \equiv_{str} u'$.*
2. *Substitutivity: if $t \equiv_{str} u$ then $t\{x \leftarrow v\} \equiv_{str} u\{x \leftarrow v\}$ for all values v .*

Theorem

1. *Net bisimilarity is compatible.*
2. *Net bisimilarity is included in contextual equivalence.*

Actually, we prove compatibility for any abstract \equiv_M which strongly commutes with \rightarrow_{vsc} and is substitutive.

Net Bisimilarity is Compatible

Proof

Lemma

Structural equivalence \equiv_{str} verifies:

1. *Strong commutation: if $t \equiv_{str} u$ and $t \rightarrow_{vsc} t'$ then $u \rightarrow_{vsc} u'$ and $t' \equiv_{str} u'$.*
2. *Substitutivity: if $t \equiv_{str} u$ then $t\{x \leftarrow v\} \equiv_{str} u\{x \leftarrow v\}$ for all values v .*

Theorem

1. *Net bisimilarity is compatible.*
2. *Net bisimilarity is included in contextual equivalence.*

Actually, we prove compatibility for any abstract \equiv_M which strongly commutes with \rightarrow_{vsc} and is substitutive.

\equiv_M -mirrored normal form bisimilarity

A relation \mathcal{R} is a \equiv_M -mirrored normal form bisimulation if, whenever $t \mathcal{R} t'$ then one of the following cases hold:

$$\text{(nai 1)} \quad t \Downarrow_{\text{vsc}} \quad \text{and} \quad t' \Downarrow_{\text{vsc}}$$

$$\text{(nai 2)} \quad t \Downarrow_{\text{vsc}} x \quad \text{and} \quad t' \Downarrow_{\text{vsc}} x$$

$$\text{(nai 3)} \quad t \Downarrow_{\text{vsc}} \lambda x. t_1 \quad \text{and} \quad t' \Downarrow_{\text{vsc}} \lambda x. t'_1 \\ \text{with } t_1 \mathcal{R} t'_1$$

$$\text{(nai 4)} \quad t \Downarrow_{\text{vsc}} n_1 n_2 \quad \text{and} \quad t' \Downarrow_{\text{vsc}} n' \equiv_M n'_1 n'_2 \\ \text{with } n_1 \mathcal{R} n'_1 \text{ and } n_2 \mathcal{R} n'_2$$

$$\text{(nai 5)} \quad t \Downarrow_{\text{vsc}} n_1 [x \leftarrow n_2] \quad \text{and} \quad t' \Downarrow_{\text{vsc}} n' \equiv_M n'_1 [x \leftarrow n'_2] \\ \text{with } n_1 \mathcal{R} n'_1 \text{ and } n_2 \mathcal{R} n'_2$$

\equiv_M -mirrored normal form bisimilarity is defined by co-induction, as the largest \equiv_M -mirrored bisimulation.

Net bisimilarities

Examples of bisimilar terms

Net bisimilarity extends \simeq_{nai} bisimilarity, and identifies meaningless terms.

- ▶ (meaningful) $Y_v \simeq_{net} \Theta_v$

As did Lassen, we manage to equate the fixed point combinators.

- ▶ (meaningless)

While Lassen differentiated between meaningless terms, net bisimilarity equates them.

- ▶ (meaningless left) $(xx)(\delta\delta) \simeq_{net} \delta\delta$

- ▶ (meaningless) $\Omega_L \simeq_{net} \Omega_R \simeq_{net} \Omega = \delta\delta$

- ▶ (proof nets) Net bisimilarity validates structural equivalence.

$$xI[x \leftarrow yI] \simeq_{net} x[x \leftarrow yI]I$$

Net bisimilarity

Shortcomings

Net bisimilarity does not subsume Lassen's enf bisimilarity, and is still far away from full abstraction.

▶ (id) $(\lambda x.x)(yy) \not\equiv_{net} yy$

While this equation is validated by enf bisimilarity as part of Moggi's equations.

▶ (duplication) $(xx)[x \leftarrow yy] \not\equiv_{net} (yy)(yy)$

As for enf, duplication is not accounted for by net bisimilarity.

Net Bisimilarity and the Identity Rule

A first step to be able to include Lassen's Enf in Net Bisimilarity is to include the identity rule ($\text{It} \equiv_{\text{lid}} t$).

Current technique to add \equiv_{str} does not adapt:

contextual \equiv_{lid} does not strongly commute with \rightarrow_{vsc}

$$\begin{array}{ccc} yv[y \leftarrow x[x \leftarrow t]] & \equiv_{\text{lid}} & yv[y \leftarrow t] \\ \downarrow_{\text{vsc}} & & \downarrow_{\text{vsc}} \\ xv[x \leftarrow t] & = & yv[y \leftarrow t] \end{array}$$

On the other hand, improving enf to match net is not easy.

It is not even easy to define enf with weak reduction instead of left to right.

Enf and Net bisimilarities are orthogonal, and there does not seem to be an easy way to mix them.

Outline

Programming Languages & Program Equivalence

Equivalence of Programs

Plotkin's Style Call-by-Value

Lassen's Call-by-Value Bisimilarity

An Open Call-by-Value Calculus: The Value Substitution Calculus

(New) Normal Form Bisimilarities

Type Equivalence

Conclusion and Discussion

From Operational to Denotational

Relational Semantics

We investigate [Ehrhard's CbV relational model](#), which is not fully abstract for contextual equivalence, as it does not satisfy duplication.

The model induces an [equational theory](#) on terms (identifying terms with the same interpretation).

This equational theory does not have a **syntactic characterization** but it can still be studied via [non idempotent intersection types](#).

Multi Types by Value

"Typing" system

LINEAR TYPES $L, L' ::= M \multimap N$
MULTI TYPES $M, N ::= [L_1, \dots, L_n] \quad n \geq 0$

$$\frac{}{x : [L] \vdash x : L} \text{ax} \qquad \frac{\Gamma, x : M \vdash t : N}{\Gamma \vdash \lambda x. t : M \multimap N} \lambda$$
$$\frac{\Gamma \vdash t : [M \multimap N] \quad \Delta \vdash u : M}{\Gamma \uplus \Delta \vdash tu : N} \text{@} \qquad \frac{\Gamma, x : M \vdash t : N \quad \Delta \vdash u : M}{\Gamma \uplus \Delta \vdash t[x \leftarrow u] : N} \text{es}$$
$$\frac{(\Gamma_i \vdash v : L_i)_{i \in I} \quad I \text{ finite}}{\uplus_{i \in I} \Gamma_i \vdash v : \uplus_{i \in I} [L_i]} \text{many}$$

Figure: Call-by-Value Multi Type System for VSC.

Type Equivalence

Definition (Type equivalence)

Two terms t and t' are type equivalent, $t \simeq_{\text{type}} t'$ if:

$$\forall \Gamma, M \quad \Gamma \vdash t : M \iff \Gamma \vdash t' : M$$

Universal quantification :(

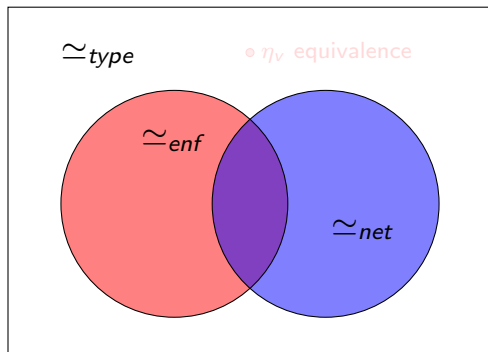
Theorem

1. *Compatibility*: if $t \simeq_{\text{type}} t'$ then, for all C , $C\langle t \rangle \simeq_{\text{type}} C\langle t' \rangle$.
2. *Soundness*: if $t \simeq_{\text{type}} t'$ then $t \simeq_C^{\text{CbV}} t'$.

Type Equivalence vs. Enf and Net

Proposition

Enf and net bisimilarities are included in Type Equivalence.

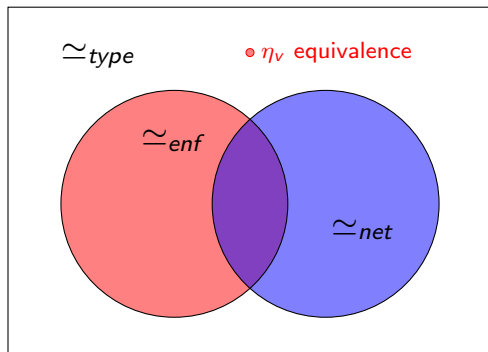


(Extensionality) $\lambda y. v y \equiv_{\eta_v} v$

Type Equivalence vs. Enf and Net

Proposition

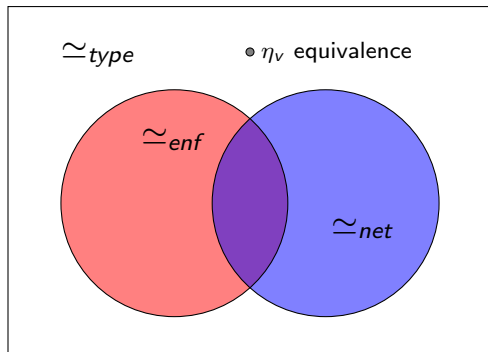
Enf and net bisimilarities are included in Type Equivalence.



(Extensionality) $\lambda y.vy \equiv_{\eta_v} v$

An axiomatisation to type equivalence?

η_v is not a problem for enf (η_v -enf already exists)!
For net, the addition of \equiv_{η_v} requires that of \equiv_{lid}



Conjecture: $\simeq_{type} = \simeq_{enf} + \simeq_{net} + \eta_v$

Outline

Programming Languages & Program Equivalence

Equivalence of Programs

Plotkin's Style Call-by-Value

Lassen's Call-by-Value Bisimilarity

An Open Call-by-Value Calculus: The Value Substitution Calculus

(New) Normal Form Bisimilarities

Type Equivalence

Conclusion and Discussion

Conclusion

Many Approaches to CbV Program Equivalence

We investigated and related three CbV normal form bisimulations and one denotational equivalence.

- ▶ Naive CbV
as an adaptation of CbN nf-bisimulations
- ▶ Lassen's Enf
state-of-the-art technique that does not comply with CbV meaninglessness
- ▶ Net, and other mirrored bisimulations
as Naive-ish bisimilarities for an extended CbV calculus - VSC
- ▶ Type Equivalence
a *universally quantified* program equivalence, that we want to axiomatize

Conclusion

A richer situation than in Call-by-Name

CbN-style approaches, even in richer settings, do not yield complete CbV normal form bisimulations. Axiomatization is harder!

Call-by-Name contextual equivalence: head normal form bisimulations up to η

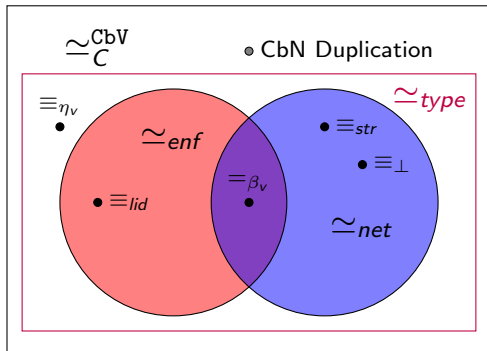
Call-by-Value contextual equivalence: Naive CbV normal form bisimulations up to η_v , **identifying meaningless**, \equiv_{lid} , \equiv_{str} , **duplication**, ...?

OR

Call-by-Value contextual equivalence: Naive CbV-**VSC** normal form bisimulations up to η_v , ~~identifying meaningless~~, \equiv_{lid} , \equiv_{str} , **duplication**, ...?

Thank you for your attention!

<https://arxiv.org/abs/2303.08161>



- \equiv_{\perp} : identifying meaningless terms
- \equiv_{lid} : Moggi's identity rule $!t \equiv_{\text{lid}} t$
- \equiv_{str} : structural equivalence
- $=_{\beta_v}$: β_v -conversion
- \equiv_{η_v} : η_v -equivalence $\lambda x.yx \equiv_{\eta_v} y$



Beniamino Accattoli and Giulio Guerrieri.

Open call-by-value.

In Atsushi Igarashi, editor, *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 206–226, 2016.



Beniamino Accattoli and Giulio Guerrieri.

The theory of call-by-value solvability (long version).

CoRR, abs/2207.08697, 2022.



Beniamino Accattoli and Luca Paolini.

Call-by-value solvability, revisited.

In Tom Schrijvers and Peter Thiemann, editors, *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, volume 7294 of *Lecture Notes in Computer Science*, pages 4–16. Springer, 2012.



Adrien Durier, Daniel Hirschhoff, and Davide Sangiorgi.

Eager functions as processes.

Theor. Comput. Sci., 913:8–42, 2022.



Søren B Lassen.

Bisimulation in untyped lambda calculus:: Böhm trees and bisimulation up to context.

Electronic Notes in Theoretical Computer Science,
20:346–374, 1999.



Soren Lassen.

Eager normal form bisimulation.

In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science, LICS '05*, page 345–354, USA, 2005.
IEEE Computer Society.



Eugenio Moggi.

Computational λ -Calculus and Monads.

In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89)*, Pacific Grove, California, USA, June 5-8, 1989, pages 14–23. IEEE Computer Society, 1989.



G.D. Plotkin.

Call-by-name, call-by-value and the λ -calculus.

Theoretical Computer Science, 1(2):125–159, 1975.



D. Sangiorgi.

The lazy lambda calculus in a concurrency scenario.

Information and Computation, 111(1):120–153, 1994.