Normal Form Bisimulations by Value

Beniamino Accattoli[1], Adrienne Lancelot[1] & Claudia Faggian[2]

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

May 11th 2023 - Chocola

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 functional programming languages.

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

In particular, we focus on program equivalence.

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 considered closed.

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 [Plo75].

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**.
  Convoluted definition...

In Call-by-Name, meaningful = solvable (head normalizable).
In Plotkin's Call-by-Value, meaningful $\neq$??-normalizable:

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

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	$t \mathrel{\mathcal{R}} u$	$\Rightarrow$	$C\langle t\rangle \mathrel{\mathcal{R}} C\langle u\rangle$
Adequacy	$t \mathrel{\mathcal{R}} u$	$\Rightarrow$	$t \Downarrow$ iff $u \Downarrow$
Conversion	$t =_\beta u$	$\Rightarrow$	$t \mathrel{\mathcal{R}} u$

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 a 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 ($\mathtt{I}t \equiv_{lid} t$ for all t)
  $\mathtt{I}(yy) \simeq_{enf} yy$, ...
- 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].

---

[1] But 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...

  $\mathtt{I}(yy) \not\simeq_{nai} yy, \;\; \Omega_L \not\simeq_{nai} \Omega, \;\; \mathtt{I}(\mathtt{I}(yy)) \not\simeq_{nai} \mathtt{I}(yy), \ldots$

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.
  $\mathtt{I}(yy) \not\simeq_{net} (yy)$

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:

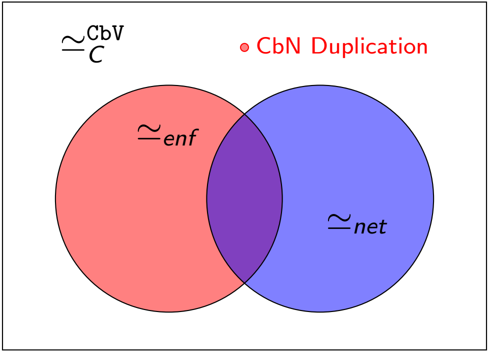
  Based on Howe's method, used in another paper [Las99] by Lassen about call-by-name normal form bisimilarities.

  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.

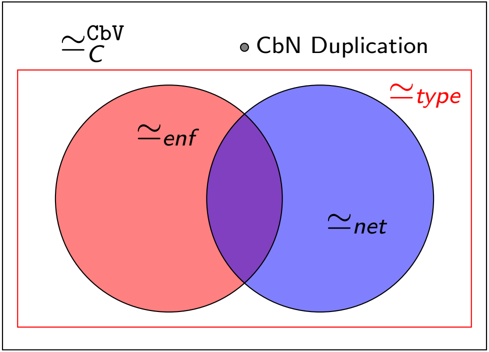


(CbN duplication) $\delta(yy) \simeq_C^{\mathtt{CbV}} (yy)(yy)$

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 $\simeq_{type}$

Contribution

Results

The two bisimilarities are orthogonal.

- Lassen's enf bisimilarity validates the identity rule.
  $\texttt{I}(t) \simeq_{enf} t$ 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**.

$$\begin{array}{llcl} \textsc{Terms} & t, u & ::= & v \mid tu \\ \textsc{Values} & v, v' & ::= & x \mid \lambda x.t \end{array}$$

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

$$\textsc{Weak Contexts}\ E ::= \langle\cdot\rangle \mid Et \mid tE$$

Weak reduction $\rightarrow_{\mathtt{w}}$ is defined by Weak contextual closure of the top-level rule $\mapsto_{\beta_v}$.

$$\begin{array}{cc} \beta_v\text{-}\textsc{reduction} & \textsc{Contextual closure} \\ (\lambda x.t)v \mapsto_{\beta_v} t\{x{\leftarrow}v\} & E\langle t\rangle \rightarrow_{\mathtt{w}} E\langle t'\rangle \quad \text{if } t \mapsto_{\beta_v} t' \end{array}$$

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 $\simeq_C^{\mathtt{CbV}}$ where the evaluation is $\Downarrow_{\mathtt{CbV}}$ associated with the weak reduction $\rightarrow_{\mathtt{w}}$.

---

[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 $t \mathrel{\mathcal{R}} t'$ 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 $t_1 \mathrel{\mathcal{R}} t_1'$

(wh 3) $t \rightarrow_{wh}^{*} x\, t_1 \ldots t_k$ and $t' \rightarrow_{wh}^{*} x\, t_1' \ldots t_k'$ with $(t_i \mathrel{\mathcal{R}} t_i')_{i \leq k}$

(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 $n \mathrel{\mathcal{R}} n'$ and $u \mathrel{\mathcal{R}} u'$

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) $t \rightarrow_{wh}^{*} x \; t_1 \ldots t_k$ and $t' \rightarrow_{wh}^{*} x \; t'_1 \ldots t'_k$ 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 $t \mathrel{\mathcal{R}} t'$ then one of the following cases hold:

(nai 1) $t \not\Downarrow_{\mathtt{w}}$ and $t' \not\Downarrow_{\mathtt{w}}$

(nai 2) $t \Downarrow_{\mathtt{w}} x$ and $t' \Downarrow_{\mathtt{w}} x$

(nai 3) $t \Downarrow_{\mathtt{w}} \lambda x.t_1$ and $t' \Downarrow_{\mathtt{w}} \lambda x.t'_1$
with $t_1 \mathrel{\mathcal{R}} t'_1$

(nai 4) $t \Downarrow_{\mathtt{w}} n_1\, n_2$ and $t' \Downarrow_{\mathtt{w}} n'_1\, n'_2$
with $n_1 \mathrel{\mathcal{R}} n'_1$ and $n_2 \mathrel{\mathcal{R}} n'_2$

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) $\mathtt{I}(yy) \not\simeq_{nai} yy$
  - (Meaningless) $\Omega_L \not\simeq_{nai} \Omega$

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 $\to_{\mathtt{las}}$.

$(xx)(\delta\delta)$ is a normal form for $\to_{\mathtt{las}}$ but $(xx)(\delta\delta) \to_{\mathtt{w}} (xx)(\delta\delta)$

Lassen differentiates between meaningless terms.

Left Normal Forms

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

$$\textsc{Left Contexts} \quad L \quad ::= \quad \langle\cdot\rangle \mid vL \mid Lt$$

$$\begin{array}{cc} \textsc{Rule at Top Level} & \textsc{Left Contextual closure} \\ (\lambda x.t)v \mapsto_{\beta_v} t\{x{\leftarrow}v\} & L\langle t\rangle \rightarrow_{\texttt{las}} L\langle u\rangle \quad \text{if } t \mapsto_{\beta_v} u \end{array}$$

Lassen defines a left normal forms grammar.

Lemma (Lassen)

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

$$\textsc{Left Normal Forms} \quad n \quad ::= \quad v \mid L\langle xv\rangle$$

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:

(enf 1) $t \not\Downarrow_{\texttt{las}}$ and $t' \not\Downarrow_{\texttt{las}}$

(enf 2) $t \Downarrow_{\texttt{las}} x$ and $t' \Downarrow_{\texttt{las}} x$

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

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

Lassen's Enf Bisimilarity

Enf bisimilarity

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

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 $y \notin \mathtt{fv}(u)$
- (let.1) $tu \equiv_{lad} (\lambda x.xu)t$ if $x \notin \mathtt{fv}(u)$
- (let.2) $vt \equiv_{rad} (\lambda x.vx)t$ if $x \notin \mathtt{fv}(v)$.

Shortcomings of enf bisimilarity

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

- *(extensionality)* $y \not\simeq_{enf} \lambda x.yx$

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

- *(meaningless left[3])* $(xx)(\delta\delta) \not\simeq_{enf} \delta\delta$

- *(meaningless)* $(\lambda x.\delta)(yy)\delta \not\simeq_{enf} \delta\delta$

---

[3] Here 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

$$\begin{array}{lrcl} \textsc{Terms} & t,u,s & ::= & v \mid tu \mid t[x{\leftarrow}u] \\ \textsc{Values} & v & ::= & x \mid \lambda x.t \end{array}$$

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 $\to_{\mathtt{m}}$ rule and the exponential $\to_{\mathtt{e}}$ rule.

Simplified Rules

$$(\lambda x.t)u \to_{\mathtt{m}} t[x{\leftarrow}u]$$
$$t[x{\leftarrow}v] \to_{\mathtt{e}} t\{x{\leftarrow}v\}$$

The β_v-reduction is fractioned: $II \to_{\beta_v} I$ but $II \to_{\mathtt{m}} x[x{\leftarrow}I] \to_{\mathtt{e}} I$

Value Substitution Calculus

Reduction at a distance

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

Substitution Contexts are lists of substitutions:

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

Actual reduction rules are at a distance.

Reduction Rules

$$S\langle\lambda x.t\rangle u \rightarrow_{\mathtt{m}} S\langle t[x{\leftarrow}u]\rangle$$

$$t[x{\leftarrow}S\langle v\rangle] \rightarrow_{\mathtt{e}} S\langle t\{x{\leftarrow}v\}\rangle$$

Value Substitution Calculus

Reduction at a distance

Distance = Needed Permutations to unstuck redexes

$$(\lambda x_1.(\lambda x.t))(yu)v \to_{\mathtt{m}} (\lambda x.t)[x_1{\leftarrow}yu]v \to_{\mathtt{m}} t[x{\leftarrow}v][x_1{\leftarrow}yu] \to_{\mathtt{e}} \ldots$$

$$(\lambda x.t)((\lambda x_2.v)(zu)) \to_{\mathtt{m}} (\lambda x.t)(v[x_2{\leftarrow}zu]) \to_{\mathtt{m}} t[x{\leftarrow}v[x_2{\leftarrow}zu]]$$
$$\to_{\mathtt{e}} t\{x{\leftarrow}v\}[x_2{\leftarrow}zu] \to_{\mathtt{vsc}} \ldots$$

Value Substitution Calculus

VSC has a powerful reduction

$$\Omega_L = (\lambda x.\delta)(yy)\delta \to_{\mathtt{m}} \delta[x{\leftarrow}yy]\delta \to_{\mathtt{m}} zz[z{\leftarrow}\delta][x{\leftarrow}yy]$$

$$\to_{\mathtt{e}} \delta\delta[x{\leftarrow}yy] \to_{\mathtt{m}}\to_{\mathtt{e}} \delta\delta[x{\leftarrow}yy] \to_{\mathtt{m}}\to_{\mathtt{e}} \ldots$$

$$\Omega_R = \delta((\lambda x.\delta)(yy)) \to_{\mathtt{m}} \delta(\delta[x{\leftarrow}yy]) \to_{\mathtt{m}} zz[z{\leftarrow}\delta[x{\leftarrow}yy]]$$

$$\to_{\mathtt{e}} \delta\delta[x{\leftarrow}yy] \to_{\mathtt{m}}\to_{\mathtt{e}} \delta\delta[x{\leftarrow}yy] \to_{\mathtt{m}}\to_{\mathtt{e}} \ldots$$

Theorem (Operational Characterization of Meaninglessness [AP12])

A term t is meaningless iff $t \not\Downarrow_{\mathtt{vsc}}$.

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:

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

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

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

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

(nai 5) $t \Downarrow_{\mathtt{vsc}} n_1[x{\leftarrow}n_2]$ and $t' \Downarrow_{\mathtt{vsc}} n'_1[x{\leftarrow}n'_2]$
with $n_1 \; \mathcal{R} \; n'_1$ 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
    $x[x{\leftarrow}t] \not\simeq_{nai} t$
  - And does not allow permutation between applications and explicit substitutions
    $x\mathtt{I}[x{\leftarrow}y\mathtt{I}] \not\simeq_{nai} x[x{\leftarrow}y\mathtt{I}]\mathtt{I}$

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:

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

[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:

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

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:

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

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

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

(nai 4) $t \Downarrow_{\mathtt{vsc}} n_1 n_2$ and $t' \Downarrow_{\mathtt{vsc}} n' \equiv_{str} n_1' n_2'$
with $n_1 \; \mathcal{R} \; n_1'$ and $n_2 \; \mathcal{R} \; n_2'$

(nai 5) $t \Downarrow_{\mathtt{vsc}} n_1[x{\leftarrow}n_2]$ and $t' \Downarrow_{\mathtt{vsc}} n' \equiv_{str} n_1'[x{\leftarrow}n_2']$
with $n_1 \; \mathcal{R} \; n_1'$ 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 $t \rightarrow_{\mathtt{vsc}} t'$ then $u \rightarrow_{\mathtt{vsc}} u'$ 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 $\rightarrow_{\mathtt{vsc}}$ and is substitutive.

Net Bisimilarity is Compatible

Proof

Lemma

Structural equivalence $\equiv_{str}$ verifies:

1. *Strong commutation: if $t \equiv_{str} u$ and $t \rightarrow_{\mathtt{vsc}} t'$ then $u \rightarrow_{\mathtt{vsc}} u'$ 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 $\rightarrow_{\mathtt{vsc}}$ and is substitutive.

$\equiv_M$-mirrored normal form bisimilarity

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

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

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

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

(nai 4) $t \Downarrow_{\mathtt{vsc}} n_1 n_2$ and $t' \Downarrow_{\mathtt{vsc}} n' \equiv_M n_1' n_2'$
with $n_1 \mathrel{\mathcal{R}} n_1'$ and $n_2 \mathrel{\mathcal{R}} n_2'$

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

$\equiv_M$-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.
  $x\mathtt{I}[x{\leftarrow}y\mathtt{I}] \simeq_{net} x[x{\leftarrow}y\mathtt{I}]\mathtt{I}$

Net bisimilarity

Shortcomings

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

- (id) $(\lambda x.x)(yy) \neq_{net} yy$

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

- (duplication) $(xx)[x{\leftarrow}yy] \neq_{net} (yy)(yy)$

  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 ($\mathtt{I}t \equiv_{lid} t$).

Current technique to add $\equiv_{str}$ does not adapt:
contextual $\equiv_{lid}$ does not strongly commute with $\rightarrow_{\mathtt{vsc}}$

$$\begin{tikzcd} yv[y\leftarrow x[x\leftarrow t]] \arrow[r, phantom, "\equiv_{lid}"] \arrow[d, "\mathtt{vsc}"] & yv[y\leftarrow t] \arrow[d, "\mathtt{vsc}", "/" marking] \\ xv[x\leftarrow t] \arrow[r, phantom, "="] & yv[y\leftarrow t] \end{tikzcd}$$

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

$$\begin{array}{llll} \textsc{Linear Types} & L, L' & ::= & M \multimap N \\ \textsc{Multi Types} & M, N & ::= & [L_1, \ldots, L_n] \quad n \geq 0 \end{array}$$

$$\frac{}{x : [L] \vdash x : L}\ \mathsf{ax} \qquad \frac{\Gamma, x : M \vdash t : N}{\Gamma \vdash \lambda x.t : M \multimap N}\ \lambda$$

$$\frac{\Gamma \vdash t : [M \multimap N] \quad \Delta \vdash u : M}{\Gamma \uplus \Delta \vdash tu : N}\ \mathtt{@} \qquad \frac{\Gamma, x : M \vdash t : N \quad \Delta \vdash u : M}{\Gamma \uplus \Delta \vdash t[x{\leftarrow}u] : N}\ \mathsf{es}$$

$$\frac{(\Gamma_i \vdash v : L_i)_{i \in I} \quad I \text{ finite}}{\biguplus_{i \in I} \Gamma_i \vdash v : \biguplus_{i \in I} [L_i]}\ \mathsf{many}$$

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

Type Equivalence

Definition (Type equivalence)

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

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

Universal quantification :(

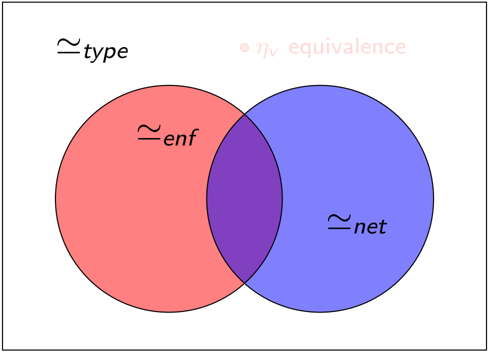
Theorem

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

Type Equivalence vs. Enf and Net

Proposition

Enf and net bisimilarities are included in Type Equivalence.

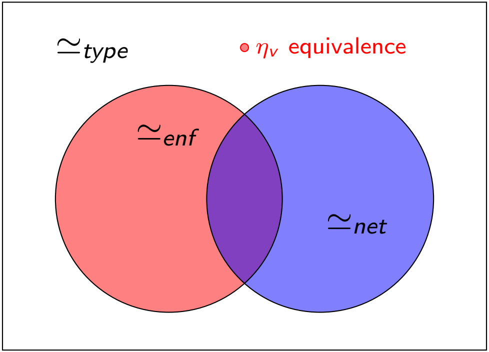


(Extensionality) $\lambda y.vy \equiv_{\eta_v} v$

Type Equivalence vs. Enf and Net

Proposition

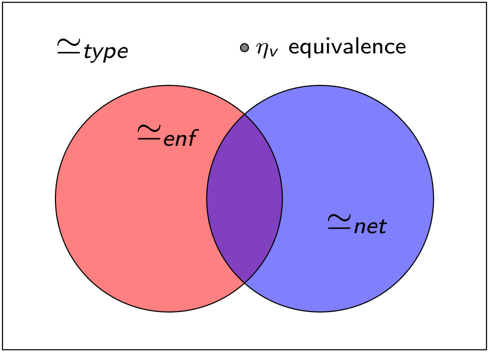
Enf and net bisimilarities are included in Type Equivalence.



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

An axiomatisation to type equivalence?

η_v is less of a problem: It is known how to extend enf (and we plan to investigate how the extension works for net!)



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

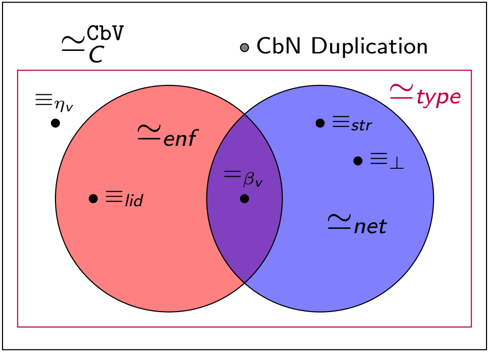
CbN-style approaches, even in richer settings, do not yield complete CbV normal form bisimulations. Axiomatization is harder!

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 $\mathtt{I}t \equiv_{lid} t$
$\equiv_{str}$: structural equivalence
$=_{\beta_v}$: β_v-conversion
$\equiv_{\eta_v}$: 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 Hirschkoff, 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.