Normal Form Bisimulations by Value

Beniamino Accattoli[1], Adrienne Lancelot[12] & Claudia Faggian[2]

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

January 14th 2024 - GALOP

Outline

Programming Languages

Program equivalence in the λ-calculus

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

Via the *untyped* 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 [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?

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:

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

Open Call-by-Value

Value Substitution Calculus & Meaningless terms

The **Value Substitution Calculus** (VSC) [AP12] is a conservative refinement of Plotkin's closed CbV based on *Linear Logic*.

Featuring explicit substitutions $[x \leftarrow t]$, it admits an operational characterization of meaningful terms:

A term t is meaningful iff $t \Downarrow_{\mathtt{vsc}}$.

Classic example solved:

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

$$\rightarrow_{\mathtt{e}} \delta\delta[x \leftarrow yy] \rightarrow_{\mathtt{m}}\rightarrow_{\mathtt{e}} \delta\delta[x \leftarrow yy] \rightarrow_{\mathtt{m}}\rightarrow_{\mathtt{e}} \dots$$

Outline

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.

  Intuitive, but the proofs are not trivial

- 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)

Lévy-Longo, Böhm and Nakajima trees have been studied in relationship with game semantics.

Ker, Nickau and Ong studied untyped CbN game semantics [KNO02]

Outline

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)
  $\mathtt{I}(yz) \simeq_{enf} yz$, ...

- 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.

[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 CbV 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 includes Linear Logic proof net equivalences.
  $t \simeq_{net} u$ if $t \equiv_{PN} u$, that is $\mathsf{ProofNet}(t) = \mathsf{ProofNet}(u)$
  Commutation: $(\lambda x.\lambda y.t)\, u\, s \equiv_{PN} (\lambda y.\lambda x.t)\, s\, u$
- It does not subsume Lassen's enf bisimilarity.
  $\mathtt{I}(yz) \not\simeq_{net} (yz)$

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:

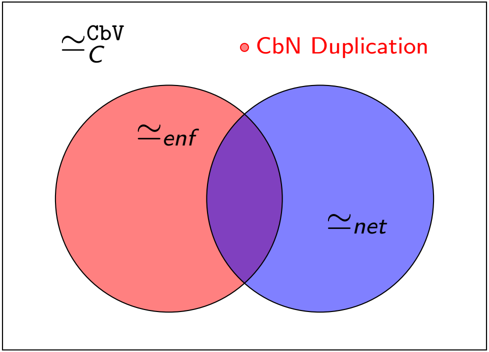
  A simpler Howe's method, used in another paper [Las99] by Lassen about call-by-name normal form bisimilarities.

  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.

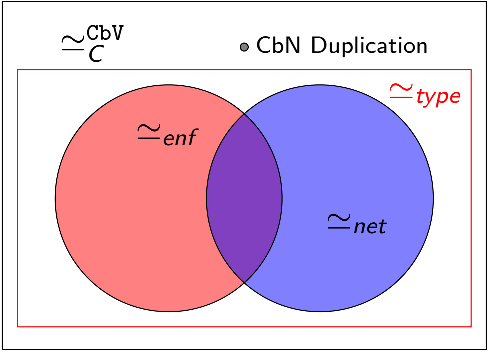


(CbN duplication) $\delta(yy) \simeq_C^{\mathtt{CbV}} (yy)(yy)$

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.

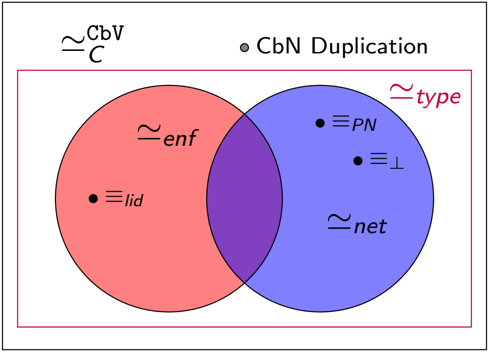


Type Equivalence $\simeq_{type}$

Contribution

Results

The two bisimilarities are orthogonal: Moggi's laws or Open Call-by-Value, but not both.

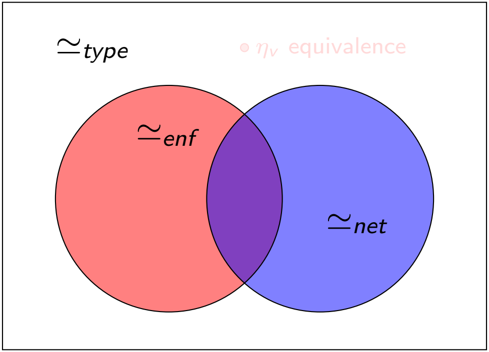


- Identity law for Lassen's Enf: $\mathtt{I}(t) \simeq_{enf} t$ for any term t
- Meaningless & Proof Nets for Net: $\Omega_L \simeq_{net} \Omega$ and $(\lambda x.\lambda y.t)(zv)(z'v') \simeq_{net} (\lambda y.\lambda x.t)(z'v')(zv)$

Zooming in on Type Equivalence

Proposition (Easier than proving Compatibility!)

Enf and net bisimilarities are included in Type Equivalence.

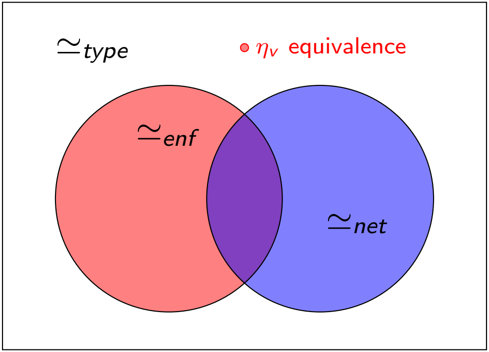


(Extensionality) $\lambda y.vy \equiv_{\eta_v} v$

Zooming in on Type Equivalence

Proposition

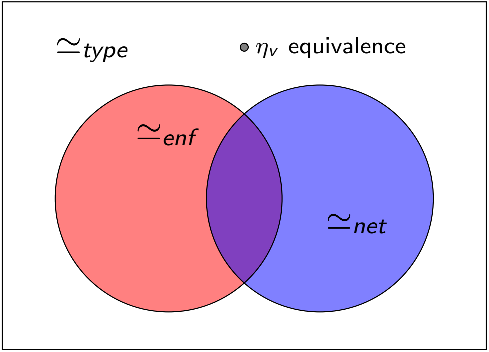
Enf and net bisimilarities are strictly included in Type Equivalence.



(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 $\equiv_{\eta_v}$ requires that of $\equiv_{lid}$

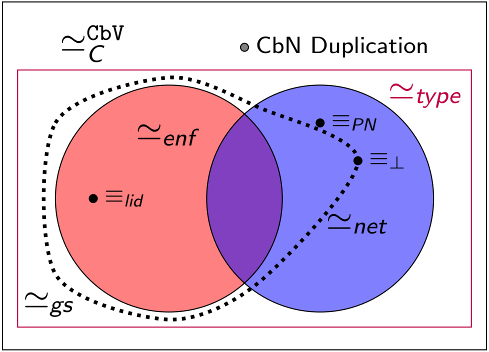


Conjecture: $\simeq_{type} = \simeq_{enf} + \simeq_{net} + \eta_v$

Outline

Where does Game Semantics fit in the picture?

How to fit 'Game Semantics' equational theory in the diagram?

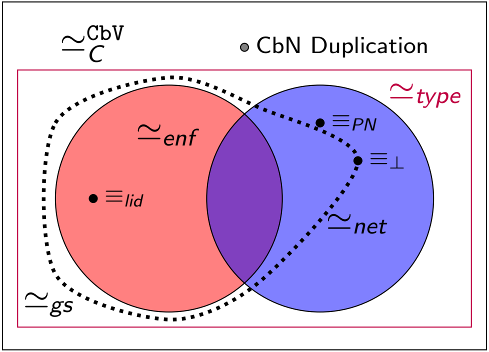


Only a guess because there are no untyped CbV game semantics

But there are untyped CbV **operational** game semantics!

Where does Game Semantics fit in the picture?

How to fit 'Game Semantics' equational theory in the diagram?



- Game semantics validate Moggi's identity law: $\mathtt{I}(xv) \simeq_{gs} xv$
- Game semantics usually do not validate commutation: $(\lambda x.\lambda y.t)(zv)(z'v') \not\simeq_{gs} (\lambda y.\lambda x.t)(z'v')(zv)$
- Any respectable model should identify meaningless terms $\equiv_{\perp}$
- Games usually do not validate duplication (of resources)

Operational Game Semantics & Normal Form Bisimulations

Operational Game Semantics are presented as bisimilarities on labelled transition systems (that do different actions depending on the shape of normal forms)

That is, normal form bisimulations but with more structure!

Surprisingly they use a different proof method for compatibility

I've never seen Lassen's/Howe's method applied to OGS!

- Enf & OGS: tried and true (many improvements and Lassen's enf is the basis of next talk's OGS)
- Net & OGS: *we do not know ...*

Outline

Programming Languages & Program Equivalence

Equivalence of Programs

Normal Form Bisimulations by Value

(Operational) Game Semantics

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

  as Naive-ish bisimilarities for an extended CbV calculus - VSC
- Type Equivalence

  a *universally quantified* program equivalence, that we want to axiomatize
- [To Do] (Operational) Game Semantics?

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

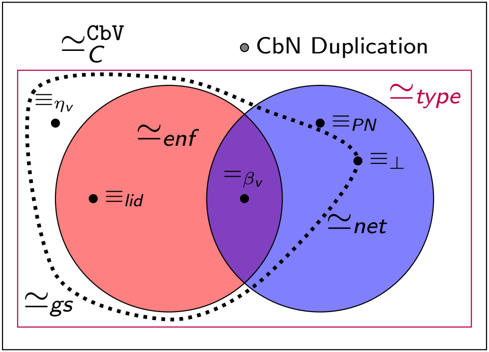
Call-by-Value contextual equivalence: Naive CbV normal form bisimulations up to η_v, identifying meaningless, $\equiv_{lid}$, $\equiv_{PN}$, duplication, ...?

OR

Call-by-Value contextual equivalence: Net bisimulations up to η_v, ~~identifying meaningless~~, $\equiv_{lid}$, ~~$\equiv_{PN}$~~, duplication, ...?

Thank you for your attention!

https://arxiv.org/abs/2303.08161



$\equiv_\bot$: identifying meaningless terms

$\equiv_{lid}$: Moggi's identity rule $\mathtt{I}t \equiv_{lid} t$

$\equiv_{PN}$: proof net 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 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.

Andrew D Ker, Hanno Nickau, and C.-H. Luke Ong.
Innocent game models of untyped -calculus.
Theoretical Computer Science, 2002.

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.

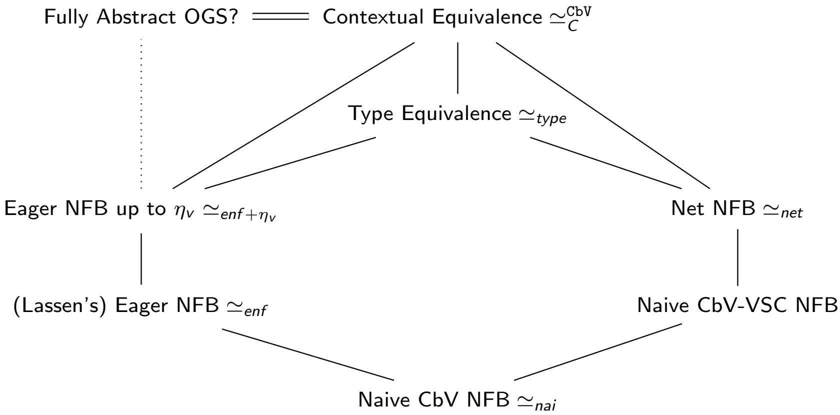
Eugenio Moggi.
Computational λ-Calculus and Monads.
In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 14–23. IEEE Computer Society, 1989.

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.

A Family of Normal Form Bisimulations by Value



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:

(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 $t_1 \; \mathcal{R} \; t_1'$

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

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

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'

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

Naive CbV-VSC normal form bisimilarity

A relation $\mathcal{R}$ is a naive CbV-VSC 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 $t' \Downarrow_{\mathtt{vsc}} n'_1 n'_2$
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 $t' \Downarrow_{\mathtt{vsc}} n'_1[x{\leftarrow}n'_2]$
with $n_1 \mathrel{\mathcal{R}} n'_1$ and $n_2 \mathrel{\mathcal{R}} n'_2$

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

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:

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

2

[2] $\equiv_{str}$ is the smallest equivalence relation, which includes these equalities and that is compatible.

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 $\mathrm{ProofNet}(t) = \mathrm{ProofNet}(u)$

Structural Equivalence is equivalent to a syntactic axiomatization:

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

[2]

[2] $\equiv_{str}$ 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 $t \mathrel{\mathcal{R}} t'$ 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 $t' \Downarrow_{\mathtt{vsc}} n' \equiv_{str} n'_1 n'_2$
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 $t' \Downarrow_{\mathtt{vsc}} n' \equiv_{str} n'_1[x \leftarrow n'_2]$
with $n_1 \mathrel{\mathcal{R}} n'_1$ and $n_2 \mathrel{\mathcal{R}} n'_2$

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

$\equiv_M$-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) $t \not\Downarrow_{\texttt{vsc}}$ and $t' \not\Downarrow_{\texttt{vsc}}$

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

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

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

(nai 5) $t \Downarrow_{\texttt{vsc}} n_1[x{\leftarrow}n_2]$ and $t' \Downarrow_{\texttt{vsc}} n' \equiv_M n_1'[x{\leftarrow}n_2']$
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.

Outline

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

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

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

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

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

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 :(

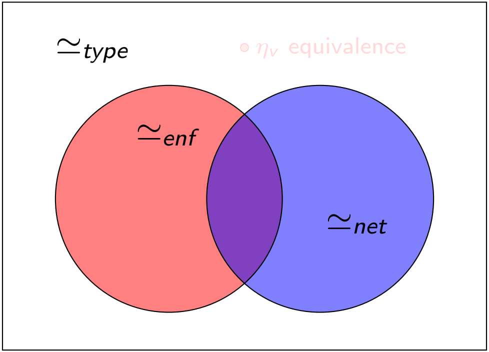
Theorem

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 $t \simeq_{C}^{\texttt{CbV}} t'$.*

Type Equivalence vs. Enf and Net

Proposition

Enf and net bisimilarities are included in Type Equivalence.

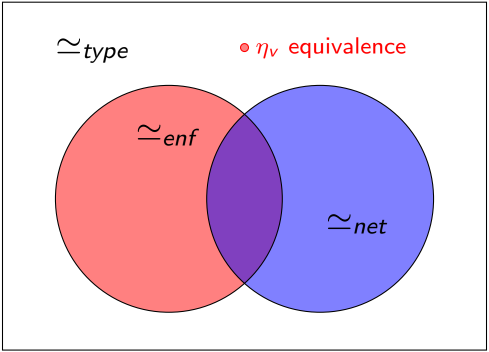


(Extensionality) $\lambda y.vy \equiv_{\eta_v} v$

Type Equivalence vs. Enf and Net

Proposition

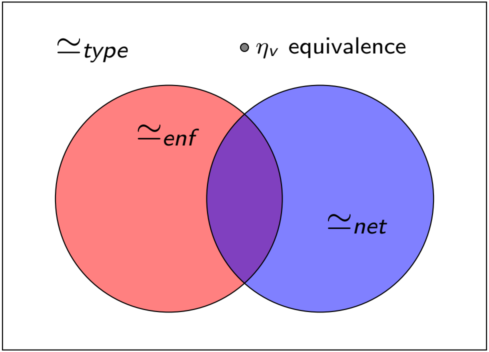
Enf and net bisimilarities are included in Type Equivalence.



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

An axiomatisation to type equivalence?

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



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