A Simple Formal Proof of the Genericity Lemma

Adrienne Lancelot, University of Bologna

Based on joint works with Beniamino Accattoli and Maxime Vemclefs:

- *Light Genericity*, FoSSaCS 2024
- *Barendregt's Theory of the λ-Calculus, Refreshed and Formalized*, ITP 2025
- My PhD thesis, 2025

June 25th 2026 - Chocola seminar

Partial Recursive Functions

Partial Recursive Functions model which mathematical functions are computable.

There is a natural *extensional preorder* on partial functions

$$f \leq_{\text{prf}} g \text{ if } \forall n \in \mathbb{N},\ f(n) = \bot \text{ or } f(n) =_{\mathbb{N}} g(n)$$

$f_\bot : n \mapsto \bot$ is the minimum PRF function for $\leq_{\text{prf}}$

Partial Recursive Functions

Partial Recursive Functions model which mathematical functions are computable.

There is a natural *extensional preorder* on partial functions

$$f \leq_{\mathsf{prf}} g \text{ if } \forall n \in \mathbb{N},\ f(n) = \bot \text{ or } f(n) =_{\mathbb{N}} g(n)$$

$f_\bot : n \mapsto \bot$ is the minimum PRF function for $\leq_{\mathsf{prf}}$

Partial Recursive Functions

Partial Recursive Functions model which mathematical functions are computable.

There is a natural *extensional preorder* on partial functions

$$f \leq_{\text{prf}} g \text{ if } \forall n \in \mathbb{N},\ f(n) = \bot \text{ or } f(n) =_{\mathbb{N}} g(n)$$

$f_\bot : n \mapsto \bot$ is the minimum PRF function for $\leq_{\text{prf}}$

Lambda Calculus

PRF do not look at how to compute, hence the preorder can only be extensional.

Instead, in the lambda calculus, how to compute is a critical concept.

There are a rich number of possible equivalences (or preorders) of lambda terms, both extensional or intensional.

Lambda Calculus

PRF do not look at how to compute, hence the preorder can only be extensional.

Instead, in the lambda calculus, how to compute is a critical concept.

There are a rich number of possible equivalences (or preorders) of lambda terms, both extensional or intensional.

Lambda Calculus

PRF do not look at how to compute, hence the preorder can only be extensional.

Instead, in the lambda calculus, how to compute is a critical concept.

There are a rich number of possible equivalences (or preorders) of lambda terms, both extensional or intensional.

Computable Functions & Lambda Calculus

Partial recursive functions embed in the lambda calculus.

What is the lambda term that represents **undefined**?

A computation that never ends? Ω!

$\Omega := (\lambda x.xx)(\lambda x.xx) \rightarrow_\beta (\lambda x.xx)(\lambda x.xx) \rightarrow_\beta \cdots$

Now, what is the equivalence class of **undefined**/Ω ?

Computable Functions & Lambda Calculus

Partial recursive functions embed in the lambda calculus.

What is the lambda term that represents **undefined**?

A computation that never ends? Ω!

$\Omega := (\lambda x.xx)(\lambda x.xx) \rightarrow_\beta (\lambda x.xx)(\lambda x.xx) \rightarrow_\beta \cdots$

Now, what is the equivalence class of **undefined**/Ω ?

Computable Functions & Lambda Calculus

Partial recursive functions embed in the lambda calculus.

What is the lambda term that represents **undefined**?

A computation that never ends? Ω!

$\Omega := (\lambda x.xx)(\lambda x.xx) \rightarrow_\beta (\lambda x.xx)(\lambda x.xx) \rightarrow_\beta \cdots$

Now, what is the equivalence class of **undefined**/Ω ?

A first naive attempt

Undefined represents a computation that never ends.

- **undefined** terms $=$ β-diverging terms?

Surprisingly, this would lead to an inconsistency.

$>>$ If all β-diverging terms are equated in an equational theory, then this theory equates all terms.

A first naive attempt

Undefined represents a computation that never ends.

- **undefined** terms = β-diverging terms?

Surprisingly, this would lead to an inconsistency.

$>>$ If all β-diverging terms are equated in an equational theory, then this theory equates all terms.

A first naive attempt

Undefined represents a computation that never ends.

- **undefined** terms = β-diverging terms?

Surprisingly, this would lead to an inconsistency.

$>>$ If all β-diverging terms are equated in an equational theory, then this theory equates all terms.

β-diverging terms may be very different

Indeed, let us look at two β-diverging terms

$$
\begin{array}{ccc}
\mathit{fix} & \text{and} & \Omega \\
\downarrow_\beta & & \downarrow_\beta \\
\lambda f.f\ (\mathit{fix}\, f) & & \Omega \\
\downarrow_\beta & & \downarrow_\beta \\
\lambda f.f\ (f\ (\mathit{fix}\, f)) & & \Omega \\
\downarrow_\beta & & \downarrow_\beta \\
\lambda f.f\ (f\ (f\ (\mathit{fix}\, f))) & & \Omega \\
\downarrow_\beta & & \downarrow_\beta \\
\lambda f.f\ (f\ (f\ (f\ \ldots))) & & \Omega \\
\downarrow_\beta & & \downarrow_\beta \\
\vdots & & \vdots
\end{array}
$$

Recursion does not carry the same meaning as looping on itself.

A second attempt

Instead, one might consider a more restrained reduction

- **undefined** terms = **head**-diverging terms?

The equational theory that identifies **head**-diverging terms is consistent.

>> This theory does not equate all terms.

A second attempt

Instead, one might consider a more restrained reduction

- **undefined** terms = **head**-diverging terms?

The equational theory that identifies **head**-diverging terms is consistent.

>> This theory does not equate all terms.

A second attempt

Instead, one might consider a more restrained reduction

- **undefined** terms = **head**-diverging terms?

The equational theory that identifies **head**-diverging terms is consistent.

>> This theory does not equate all terms.

β-diverging terms may be very different

Fixpoint combinators are **head**-normalizing.

$$\begin{array}{ccc} \mathit{fix} & \text{and} & \Omega \\ \downarrow_h & & \downarrow_h \\ \lambda f.f\ (\mathit{fix}\, f) & & \Omega \\ \not\downarrow_h & & \downarrow_h \\ & & \vdots \end{array}$$

Recursion and looping are nicely separated by **head** reduction.

Consistency

A relation $\mathcal{R} \subseteq \Lambda \times \Lambda$ is consistent if there exists $t, u \in \Lambda$ such that $(t, u) \notin \mathcal{R}$.

An equational theory is an equivalence relation $=_{\mathcal{T}}$ such that:

- *Invariance under Computation*: if $t \to_\beta u$ then $t =_{\mathcal{T}} u$
- *Stability by Contexts*: if $t =_{\mathcal{T}} u$ then $\forall C,\ C\langle t \rangle =_{\mathcal{T}} C\langle u \rangle$.

To validate the choice of undefined terms: Is there a consistent equational theory where undefined terms are collapsed?

Consistency

A relation $\mathcal{R} \subseteq \Lambda \times \Lambda$ is consistent if there exists $t, u \in \Lambda$ such that $(t, u) \notin \mathcal{R}$.

An equational theory is an equivalence relation $=_{\mathcal{T}}$ such that:

- *Invariance under Computation*: if $t \to_\beta u$ then $t =_{\mathcal{T}} u$
- *Stability by Contexts*: if $t =_{\mathcal{T}} u$ then $\forall C,\ C\langle t \rangle =_{\mathcal{T}} C\langle u \rangle$.

To validate the choice of undefined terms: Is there a consistent equational theory where undefined terms are collapsed?

Consistency

A relation $\mathcal{R} \subseteq \Lambda \times \Lambda$ is consistent if there exists $t, u \in \Lambda$ such that $(t, u) \notin \mathcal{R}$.

An equational theory is an equivalence relation $=_{\mathcal{T}}$ such that:

- *Invariance under Computation*: if $t \to_\beta u$ then $t =_{\mathcal{T}} u$
- *Stability by Contexts*: if $t =_{\mathcal{T}} u$ then $\forall C,\ C\langle t \rangle =_{\mathcal{T}} C\langle u \rangle$.

To validate the choice of undefined terms: Is there a consistent equational theory where undefined terms are collapsed?

What is Genericity?

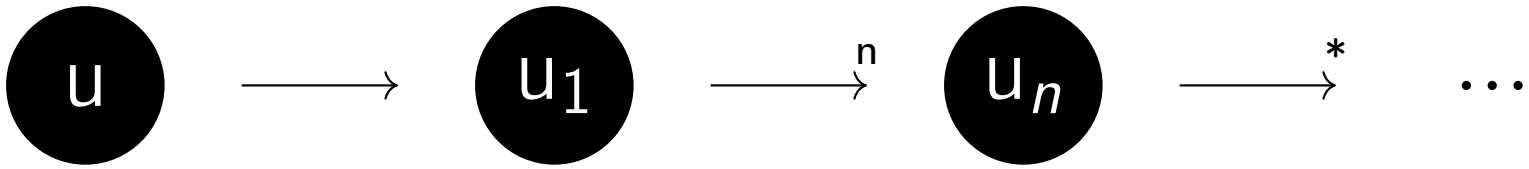
Undefined terms are black holes for the evaluation process.

If a program awaits the evaluation of an undefined sub-term

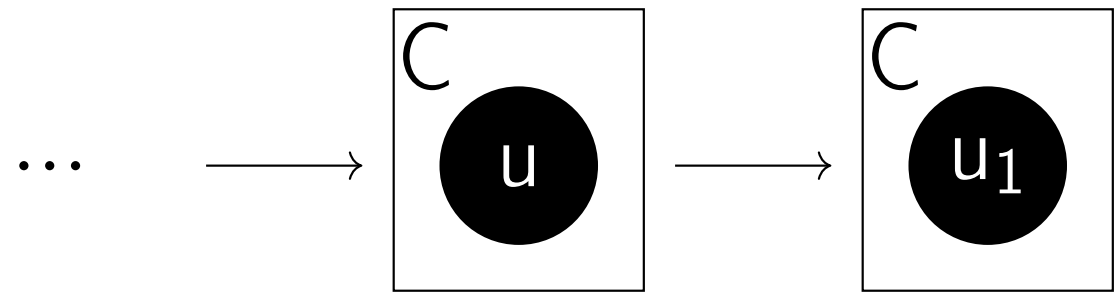
Then it will be unable to produce a result

What is Genericity?

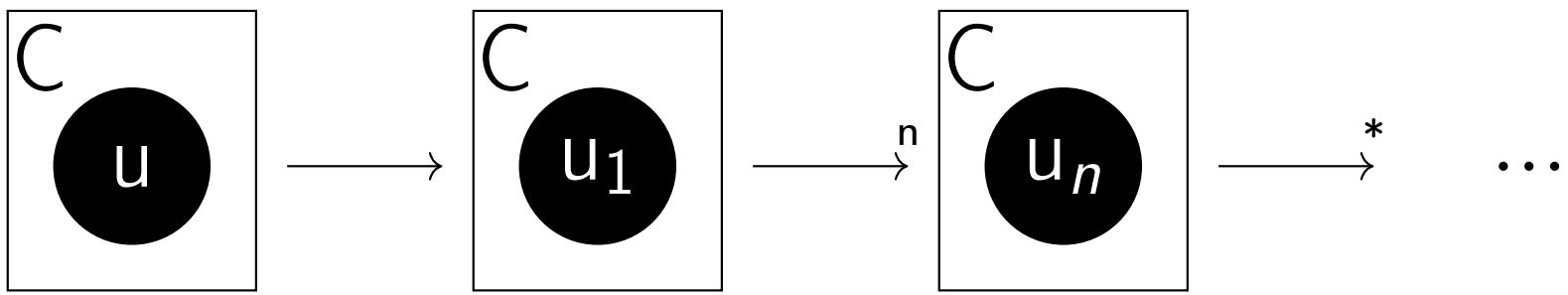
Undefined terms are black holes for the evaluation process.



If a program awaits the evaluation of an undefined sub-term



Then it will be unable to produce a result



What is Genericity?

Genericity somehow specifies this fact dually:
If a program terminates while there were **undefined** sub-terms, then it never entered the black hole.

Genericity says: (n is a normal form and s is any term)

If

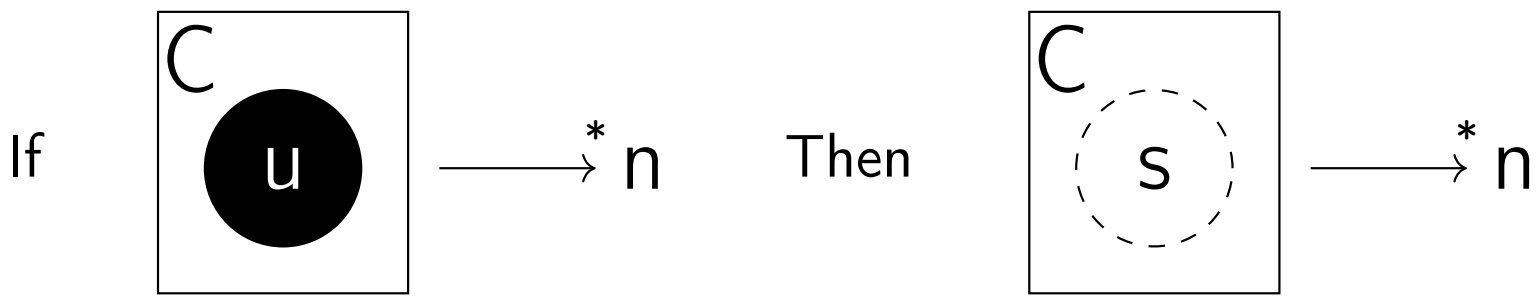
Then

Anything can simulate the *generic* **undefined** sub-terms in a terminating term.

What is Genericity?

Genericity somehow specifies this fact dually:
If a program terminates while there were **undefined** sub-terms, then it never entered the black hole.

Genericity says: (n is a normal form and s is any term)

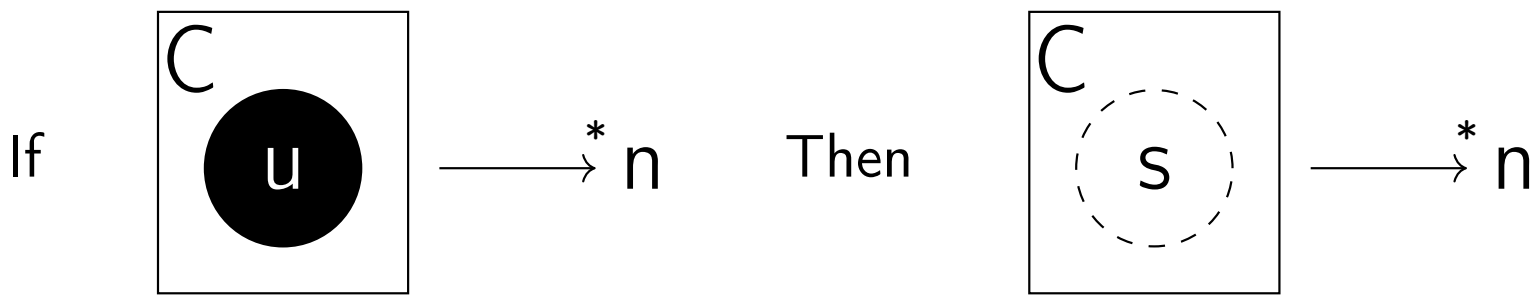


Anything can simulate the *generic* **undefined** sub-terms in a terminating term.

What is Genericity?

Genericity somehow specifies this fact dually:
If a program terminates while there were **undefined** sub-terms, then it never entered the black hole.

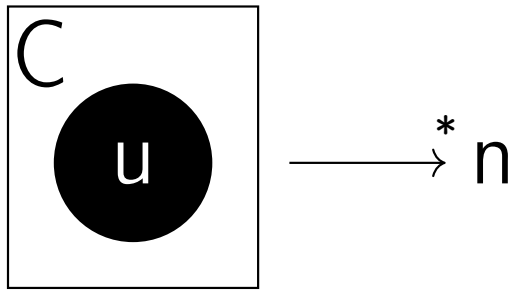
Genericity says: (n is a normal form and s is any term)

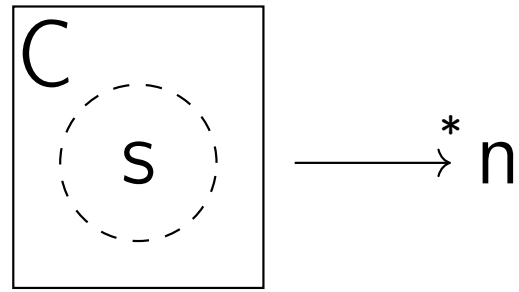


Anything can simulate the *generic* **undefined** sub-terms in a terminating term.

Genericity $\Longrightarrow$ Collapsibility

Genericity says: (n is a normal form and s is any term)

If 

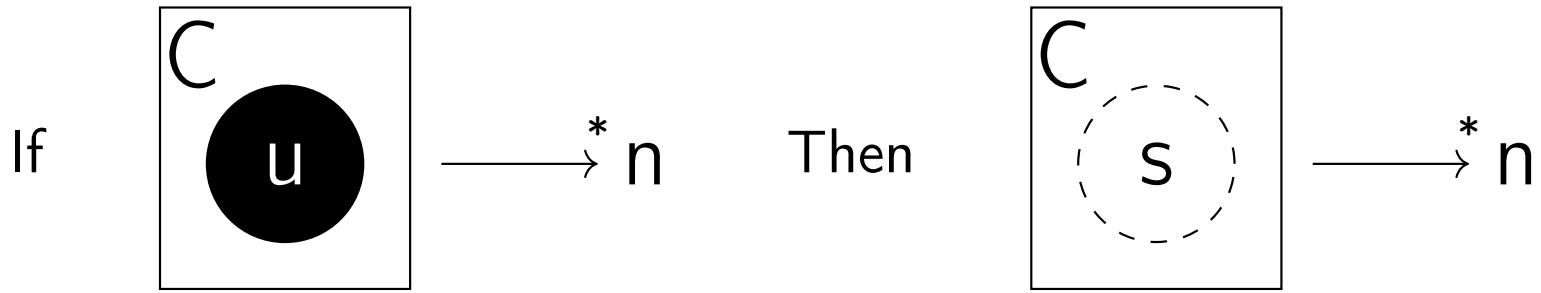
Then 

Genericity $\Longrightarrow$ Collapsibility

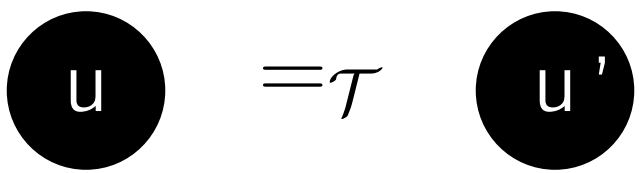
Collapsibility: there exists an equational theory $\mathcal{T}$ such that undefined terms are equated in $\mathcal{T}$ and $\mathcal{T}$ is consistent

Genericity $\Longrightarrow$ Collapsibility

Genericity says: (n is a normal form and s is any term)



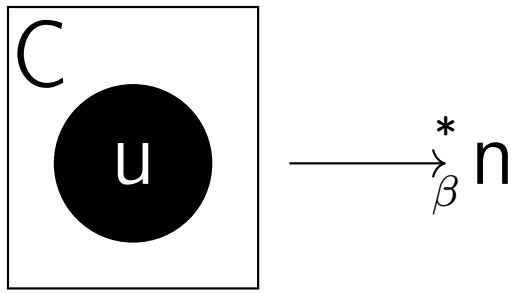
Genericity $\Longrightarrow$ Collapsibility

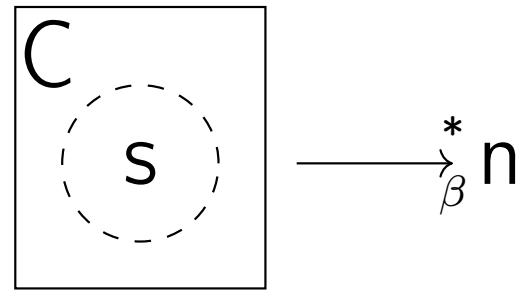


Collapsibility: there exists an equational theory $\mathcal{T}$ such that undefined terms are equated in $\mathcal{T}$ and $\mathcal{T}$ is consistent

Heavy and Light Genericity

Heavy Genericity: (n is a normal form and s is any term)

If 

Then 

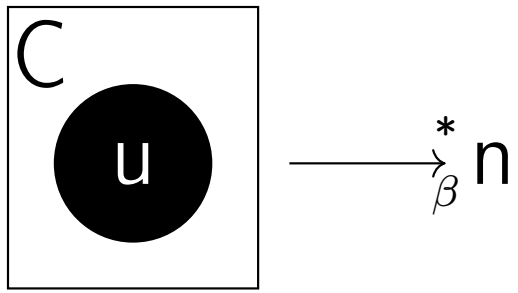
Light Genericity: (h, h' are head normal forms and s is any term)

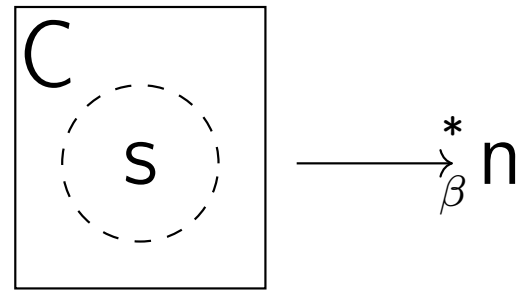
If

Then

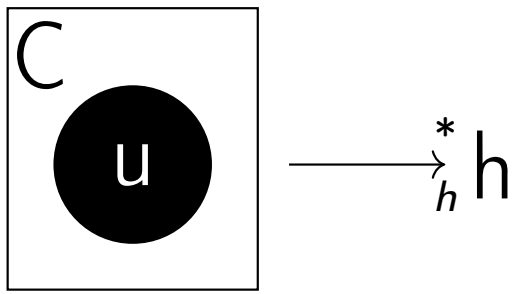
Heavy and Light Genericity

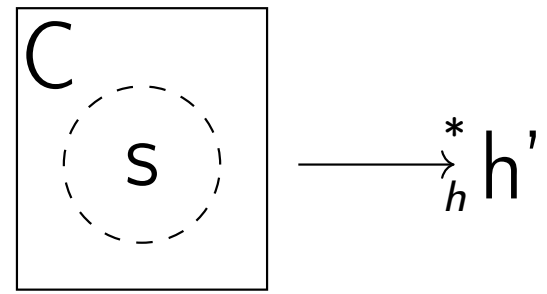
Heavy Genericity: (n is a normal form and s is any term)

If 

Then 

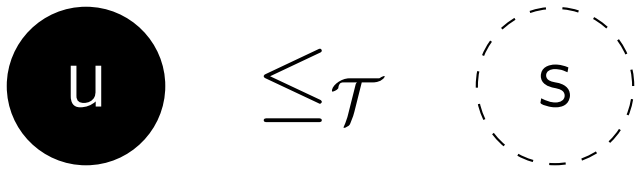
Light Genericity: (h, h' are head normal forms and s is any term)

If 

Then 

Genericity is about minimum terms

Genericity is that $\forall s$

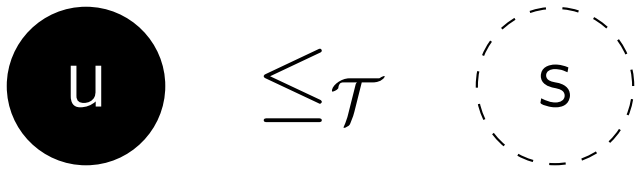


is stable by contexts

Formally, **Light Genericity** is that $\forall s$

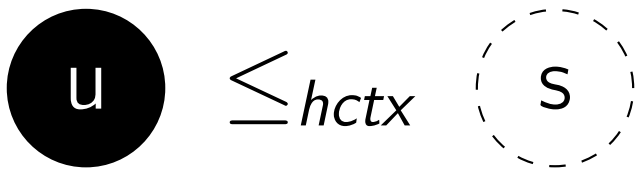
Genericity is about minimum terms

Genericity is that $\forall s$



is stable by contexts

Formally, **Light Genericity** is that $\forall s$



Refreshed Theory

What I will tell you about today:

- ... are well-known results in Call-by-Name now formalized in a proof assistant
- ... stresses that (Light) Genericity is about minimum terms and contextual preorders

What I won't tell you about today:

- Genericity results for Call-by-Value

  Accattoli-Lancelot 2024; Arrial-Guerrieri-Kesner 2024; ...
- Abstract formulation of (in)equational theories, light genericity, contextual preorder/equivalence and its maximality
- Maximum terms (co-genericity)

Refreshed Theory

What I will tell you about today:

- ... are well-known results in Call-by-Name now formalized in a proof assistant
- ... stresses that (Light) Genericity is about minimum terms and contextual preorders

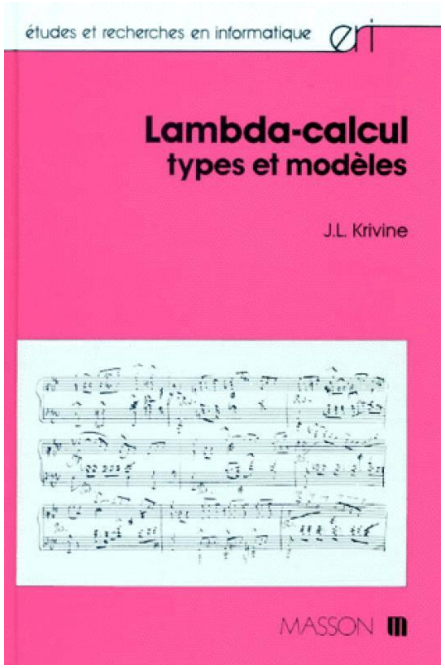
What I won't tell you about today:

- Genericity results for Call-by-Value

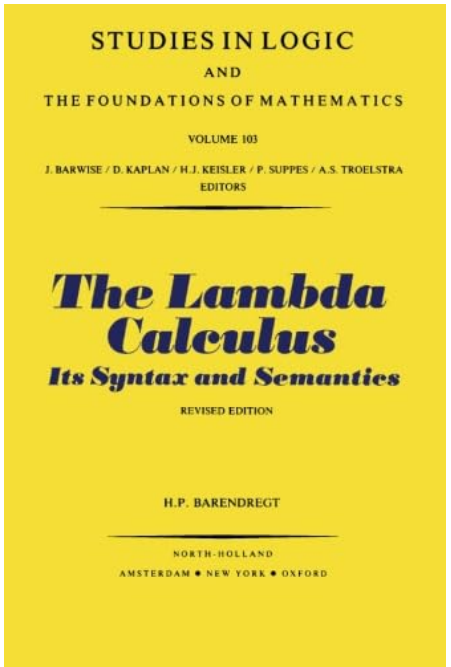
  Accattoli-Lancelot 2024; Arrial-Guerrieri-Kesner 2024; ...
- Abstract formulation of (in)equational theories, light genericity, contextual preorder/equivalence and its maximality
- Maximum terms (co-genericity)

Formalizations of the theory of the λ-calculus

- Many formal developments of the rewriting theory of the λ-calculus (confluence, etc.)

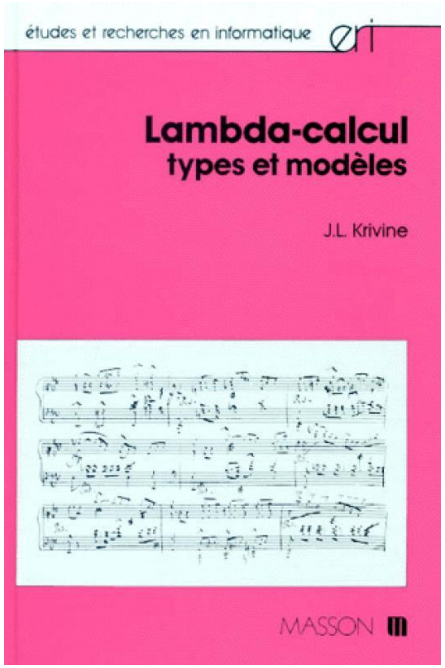


- Formalization of parts of **Krivine's book** (1990) in Rocq by Larchey-Wendling
- Formalization of $\lambda\eta$-**completeness** from Barendregt's book (1984) in HOL4 by Tian and Norrish
- Formalization of a subset of **Barendregt's book** (1984) in Abella

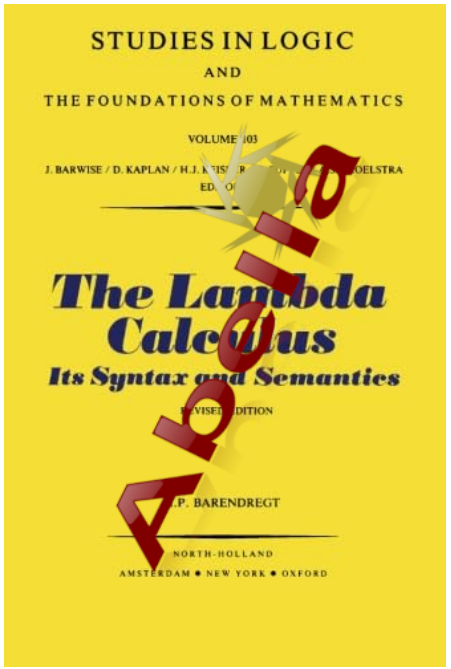


Formalizations of the theory of the λ-calculus

- Many formal developments of the rewriting theory of the λ-calculus (confluence, etc.)

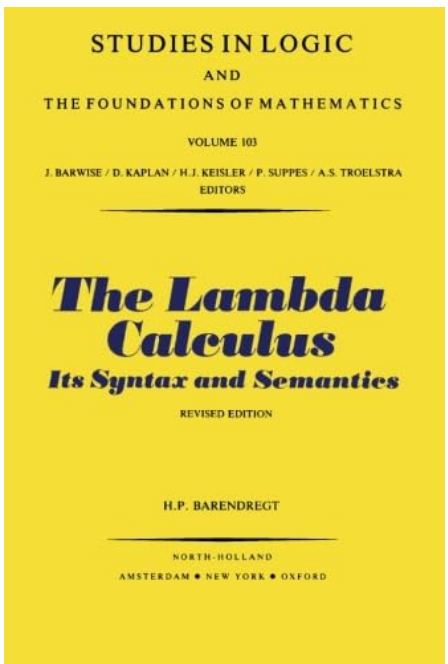


- Formalization of parts of **Krivine's book** (1990) in Rocq by Larchey-Wendling
- Formalization of $\lambda\eta$-**completeness** from Barendregt's book (1984) in HOL4 by Tian and Norrish
- Formalization of a subset of **Barendregt's book** (1984) in Abella



Refreshed and Formalized

We formalize some results of Barendregt's theory of the λ-calculus (1984).



Refreshed:

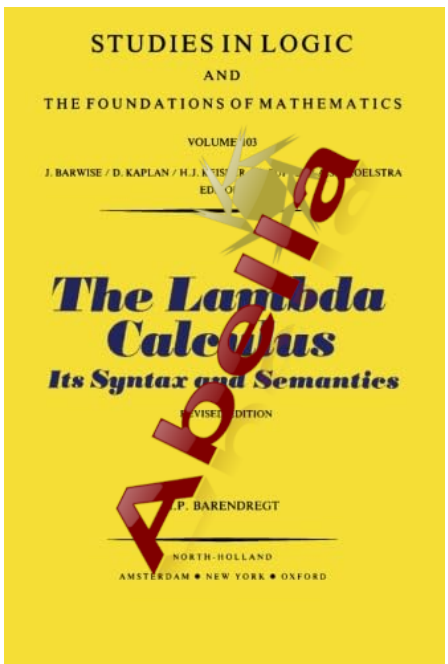
- Takahashi's proof of genericity (1994)
- Accattoli et al. study of normalization (2019)

Formalized with the Abella proof assistant:

- Reasoning with binders close to paper
- Representing contexts (with possible captures)
- Ended up leading us to new results (and new questions)

Refreshed and Formalized

We formalize some results of Barendregt's theory of the λ-calculus (1984).



Refreshed:

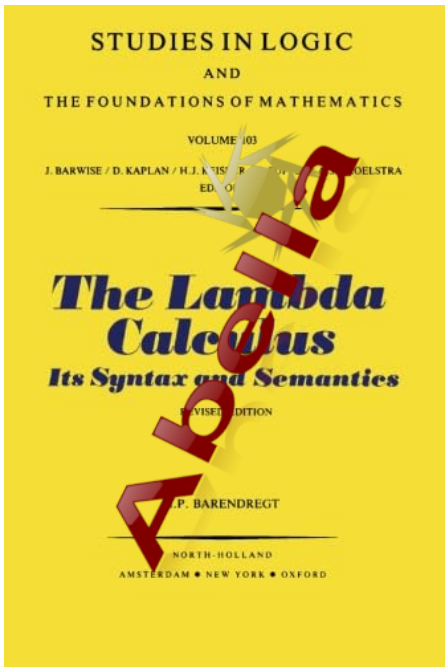
- Takahashi's proof of genericity (1994)
- Accattoli et al. study of normalization (2019)

Formalized with the Abella proof assistant:

- Reasoning with binders close to paper
- Representing contexts (with possible captures)
- Ended up leading us to new results (and new questions)

Refreshed and Formalized

We formalize some results of Barendregt's theory of the λ-calculus (1984).



Refreshed:

- Takahashi's proof of genericity (1994)
- Accattoli et al. study of normalization (2019)

Formalized with the Abella proof assistant:

- Reasoning with binders close to paper
- Representing contexts (with possible captures)
- Ended up leading us to new results (and new questions)

Proving/Formalizing Genericity

A Simple Proof of the Genericity Lemma

Masako Takahashi

Department of Information Science
Tokyo Institute of Technology
Ookayama, Meguro, Tokyo 152 Japan
masako@titisha.is.titech.ac.jp

Abstract. A short direct proof is given for the fundamental property of unsolvable λ-terms; if M is an unsolvable λ-term and $C[M]$ is solvable, then $C[N]$ is solvable for any λ-term N. (Here $C[\ ]$ stands for an arbitrary context.)

1. Preliminaries

A term in this note means a λ-term, which is either $x, \lambda x.M$ or MN, (where M, N are terms and x is a variable.) Unless otherwise stated, capital letters $M, N, P, ...$ stand for arbitrary terms, $\mathbf{M}, \mathbf{N}, ...$ for (possibly null) sequences of terms, $x, y, ...$ for variables, and $\mathbf{x}, \mathbf{y}, ...$ for (possibly null) sequences of variables. We refer to [1] as the standard text in the field.

A term of the form $\lambda \mathbf{x}.y\mathbf{M}$ (more precisely, $\lambda x_1.(\lambda x_2.(...(\lambda x_n.((...((yM_1)M_2)...)M_m))...))$ for some $n, m \geq 0$) is said to be in *head normal form* (*hnf*, for short). If a term M has a hnf (that is, $M =_\beta M'$ for a term M' in hnf), then M is called *solvable*. The following are well-known facts of solvable terms (cf.[1] 8.3.1 - 14).

(1) M is solvable if and only if $\forall P, \exists \mathbf{x}, \exists \mathbf{Q}((\lambda \mathbf{x}.M)\mathbf{Q} =_\beta P)$.
(2) $\lambda x.M$ is solvable if and only if so is M.
(3) if $M[x := N]$ is solvable then so is M.
(4) if MN is solvable then so is M.

A term in *β-normal form* (*β-nf*, for short) is recursively defined as a term of the form $\lambda \mathbf{x}.y\mathbf{M}$ where $\mathbf{M}$ is a (possibly null) sequence of terms in β-nf.

2. Propositions

First we prove a special case of the genericity lemma.

Lemma 1. Let M, N, P be terms with M unsolvable and N in β-nf. Then $P[x := M] =_\beta N$ implies $P[x := M'] =_\beta N$ for any M'.

Proof. We prove the lemma by induction on the structure of N. Suppose $P[x := M] =_\beta N$, and $N \equiv \lambda y.zN_1N_2...N_n$ where $n \geq 0$ and each N_i is in β-nf. (Here, $\equiv$ denotes syntactic equality of terms.) Then since N is solvable, P is also solvable by (3) above, and hence has a hnf, say $\lambda u.vP_1P_2...P_p$. Here x and v must be different. (For otherwise $P[x := M] =_\beta \lambda u.M\mathbf{P}$ for some $\mathbf{P}$, and $P[x := M]$ would by (2) and (4) above be unsolvable, which contradicts our assumption.) Therefore we have $P[x := M] =_\beta \lambda u.vP'_1P'_2...P'_p$ where $P'_i \equiv P_i[x := M](i = 1, 2, ..., p)$. Since $P[x := M] =_\beta N \equiv \lambda y.zN_1N_2...N_n$, we know from the Church-Rosser theorem that $P'_i =_\beta N_i (i = 1, 2, ..., n)$ and $p = n$. Without loss of generality we may also assume $\mathbf{u} \equiv \mathbf{y}$ and $v \equiv z$.

If $n = 0$, then $P =_\beta \lambda u.v \equiv N$. In this case, we have $P[x := M'] =_\beta (\lambda u.v)[x := M'] \equiv \lambda u.v \equiv N$ for any M'. If $n > 0$, then from the fact $P_i[x := M] \equiv P'_i =_\beta N_i$ and the inductive hypothesis, we get $P_i[x := M'] =_\beta N_i (i = 1, 2, ..., n)$ for any M'. In this case,

$$\begin{aligned} P[x := M'] &=_\beta (\lambda u.vP_1P_2...P_n)[x := M'] \\ &\equiv \lambda u.v(P_1[x := M'])(P_2[x := M'])...(P_n[x := M']) \\ &=_\beta \lambda y.zN_1N_2...N_n \equiv N. \end{aligned}$$

This proves the lemma. □

Lemma 2. ([1] 14.3.24. Genericity lemma) Let M be an unsolvable term, and $C[\ ]$ be a context such that $C[M]$ has a β-nf. Then $C[M] =_\beta C[M']$ for any M'.

Proof. For given M', let $\mathbf{y}$ be a sequence of all free variables in MM'. Take a new variable x (neither in $C[M]$ nor $C[M']$), and let $P \equiv C[x\mathbf{y}]$. Then since $\lambda y.M$ and $\lambda y.M'$ are closed terms, we have

$$P[x := \lambda y.M] \equiv C[(\lambda y.M)\mathbf{y}] =_\beta C[M],$$
$$P[x := \lambda y.M'] \equiv C[(\lambda y.M')\mathbf{y}] =_\beta C[M'].$$

The term $\lambda y.M$ therefore satisfies $P[x := \lambda y.M] =_\beta C[M] =_\beta N$ for some N in β-nf. Here $\lambda y.M$ is unsolvable because so is M. Hence by applying lemma 1 we get $P[x := \lambda y.M'] =_\beta N$, which implies $C[M] =_\beta C[M']$. □

Corollary 3. If M is unsolvable and $C[M]$ is solvable, then $C[M']$ is solvable for any M'.

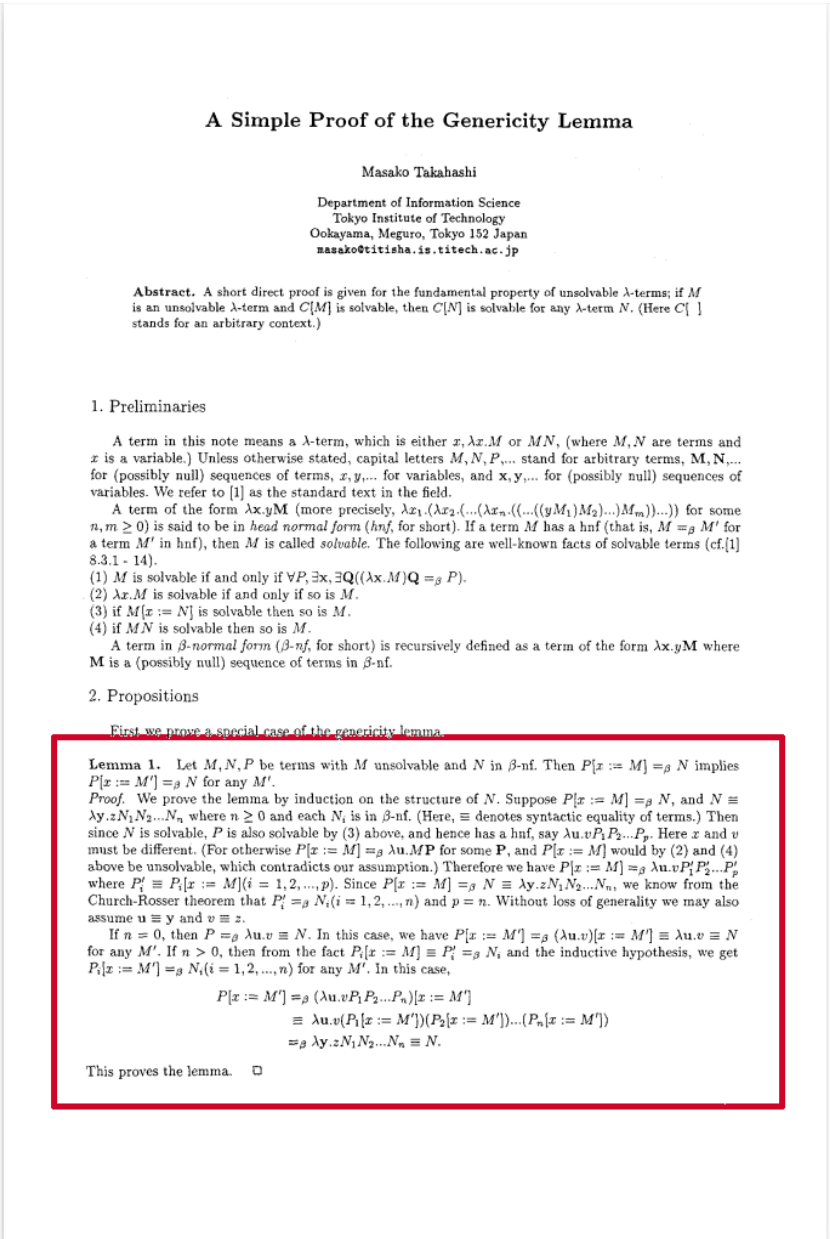
Proof. Since $C[M]$ is solvable, by (1) above there exist $\mathbf{x}$ and $\mathbf{N}$ such that $(\lambda \mathbf{x}.C[M])\mathbf{N}$ has a β-nf. Then by lemma 2 (applied to the context $(\lambda \mathbf{x}.C[\ ])\mathbf{N}$), we know $(\lambda \mathbf{x}.C[M'])\mathbf{N}$ has a β-nf for any M'. This means $(\lambda \mathbf{x}.C[M'])\mathbf{N}$ is solvable, and consequently $C[M']$ is solvable. □

The proof presented provides an alternative to the conventional one which uses a topological argument on Böhm trees (cf.[1] Chapters 10 and 14).

Reference

[1] H. P. Barendregt, *The Lambda Calculus* (North-Holland 1984).

Proving/Formalizing Genericity



A Simple Proof of the Genericity Lemma

Masako Takahashi

Department of Information Science
Tokyo Institute of Technology
Ookayama, Meguro, Tokyo 152 Japan
masako@titisha.is.titech.ac.jp

Abstract. A short direct proof is given for the fundamental property of unsolvable λ-terms; if M is an unsolvable λ-term and $C[M]$ is solvable, then $C[N]$ is solvable for any λ-term N. (Here $C[\ ]$ stands for an arbitrary context.)

1. Preliminaries

A term in this note means a λ-term, which is either $x, \lambda x.M$ or MN, (where M, N are terms and x is a variable.) Unless otherwise stated, capital letters $M, N, P, ...$ stand for arbitrary terms, $\mathbf{M}, \mathbf{N}, ...$ for (possibly null) sequences of terms, $x, y, ...$ for variables, and $\mathbf{x}, \mathbf{y}, ...$ for (possibly null) sequences of variables. We refer to [1] as the standard text in the field.

A term of the form $\lambda \mathbf{x}.y\mathbf{M}$ (more precisely, $\lambda x_1.(\lambda x_2.(...(\lambda x_n.((...((yM_1)M_2)...)M_m))...))$ for some $n, m \geq 0$) is said to be in *head normal form* (*hnf*, for short). If a term M has a hnf (that is, $M =_\beta M'$ for a term M' in hnf), then M is called *solvable*. The following are well-known facts of solvable terms (cf.[1] 8.3.1 - 14).

(1) M is solvable if and only if $\forall P, \exists \mathbf{x}, \exists \mathbf{Q}((\lambda x.M)\mathbf{Q} =_\beta P)$.
(2) $\lambda x.M$ is solvable if and only if so is M.
(3) if $M[x := N]$ is solvable then so is M.
(4) if MN is solvable then so is M.

A term in *β-normal form* (*β-nf*, for short) is recursively defined as a term of the form $\lambda \mathbf{x}.y\mathbf{M}$ where $\mathbf{M}$ is a (possibly null) sequence of terms in β-nf.

2. Propositions

First we prove a special case of the genericity lemma.

Lemma 1. Let M, N, P be terms with M unsolvable and N in β-nf. Then $P[x := M] =_\beta N$ implies $P[x := M'] =_\beta N$ for any M'.

Proof. We prove the lemma by induction on the structure of N. Suppose $P[x := M] =_\beta N$, and $N \equiv \lambda y.zN_1N_2...N_n$ where $n \geq 0$ and each N_i is in β-nf. (Here, $\equiv$ denotes syntactic equality of terms.) Then since N is solvable, P is also solvable by (3) above, and hence has a hnf, say $\lambda u.vP_1P_2...P_p$. Here x and v must be different. (For otherwise $P[x := M] =_\beta \lambda u.M\mathbf{P}$ for some $\mathbf{P}$, and $P[x := M]$ would by (2) and (4) above be unsolvable, which contradicts our assumption.) Therefore we have $P[x := M] =_\beta \lambda u.vP'_1P'_2...P'_p$ where $P'_i \equiv P_i[x := M] (i = 1, 2, ..., p)$. Since $P[x := M] =_\beta N \equiv \lambda y.zN_1N_2...N_n$, we know from the Church-Rosser theorem that $P'_i =_\beta N_i (i = 1, 2, ..., n)$ and $p = n$. Without loss of generality we may also assume $\mathbf{u} \equiv \mathbf{y}$ and $v \equiv z$.

If $n = 0$, then $P =_\beta \lambda u.v \equiv N$. In this case, we have $P[x := M'] =_\beta (\lambda u.v)[x := M'] \equiv \lambda u.v \equiv N$ for any M'. If $n > 0$, then from the fact $P_i[x := M] \equiv P'_i =_\beta N_i$ and the inductive hypothesis, we get $P_i[x := M'] =_\beta N_i (i = 1, 2, ..., n)$ for any M'. In this case,

$$\begin{aligned} P[x := M'] &=_\beta (\lambda u.vP_1P_2...P_n)[x := M'] \\ &\equiv \lambda u.v(P_1[x := M'])(P_2[x := M'])...(P_n[x := M']) \\ &=_\beta \lambda y.zN_1N_2...N_n \equiv N. \end{aligned}$$

This proves the lemma. □

Lemma 2. ([1] 14.3.24. Genericity lemma) Let M be an unsolvable term, and $C[\ ]$ be a context such that $C[M]$ has a β-nf. Then $C[M] =_\beta C[M']$ for any M'.

Proof. For given M', let $\mathbf{y}$ be a sequence of all free variables in MM'. Take a new variable x (neither in $C[M]$ nor $C[M']$), and let $P \equiv C[x\mathbf{y}]$. Then since $\lambda y.M$ and $\lambda y.M'$ are closed terms, we have

$$P[x := \lambda y.M] \equiv C[(\lambda y.M)\mathbf{y}] =_\beta C[M],$$
$$P[x := \lambda y.M'] \equiv C[(\lambda y.M')\mathbf{y}] =_\beta C[M'].$$

The term $\lambda y.M$ therefore satisfies $P[x := \lambda y.M] =_\beta C[M] =_\beta N$ for some N in β-nf. Here $\lambda y.M$ is unsolvable because so is M. Hence by applying lemma 1 we get $P[x := \lambda y.M'] =_\beta N$, which implies $C[M] =_\beta C[M']$. □

Corollary 3. If M is unsolvable and $C[M]$ is solvable, then $C[M']$ is solvable for any M'.

Proof. Since $C[M]$ is solvable, by (1) above there exist $\mathbf{x}$ and $\mathbf{N}$ such that $(\lambda x.C[M])\mathbf{N}$ has a β-nf. Then by lemma 2 (applied to the context $(\lambda x.C[\ ])\mathbf{N}$), we know $(\lambda x.C[M'])\mathbf{N}$ has a β-nf for any M'. This means $(\lambda x.C[M'])\mathbf{N}$ is solvable, and consequently $C[M']$ is solvable. □

The proof presented provides an alternative to the conventional one which uses a topological argument on Böhm trees (cf.[1] Chapters 10 and 14).

Reference

[1] H. P. Barendregt, *The Lambda Calculus* (North-Holland 1984).

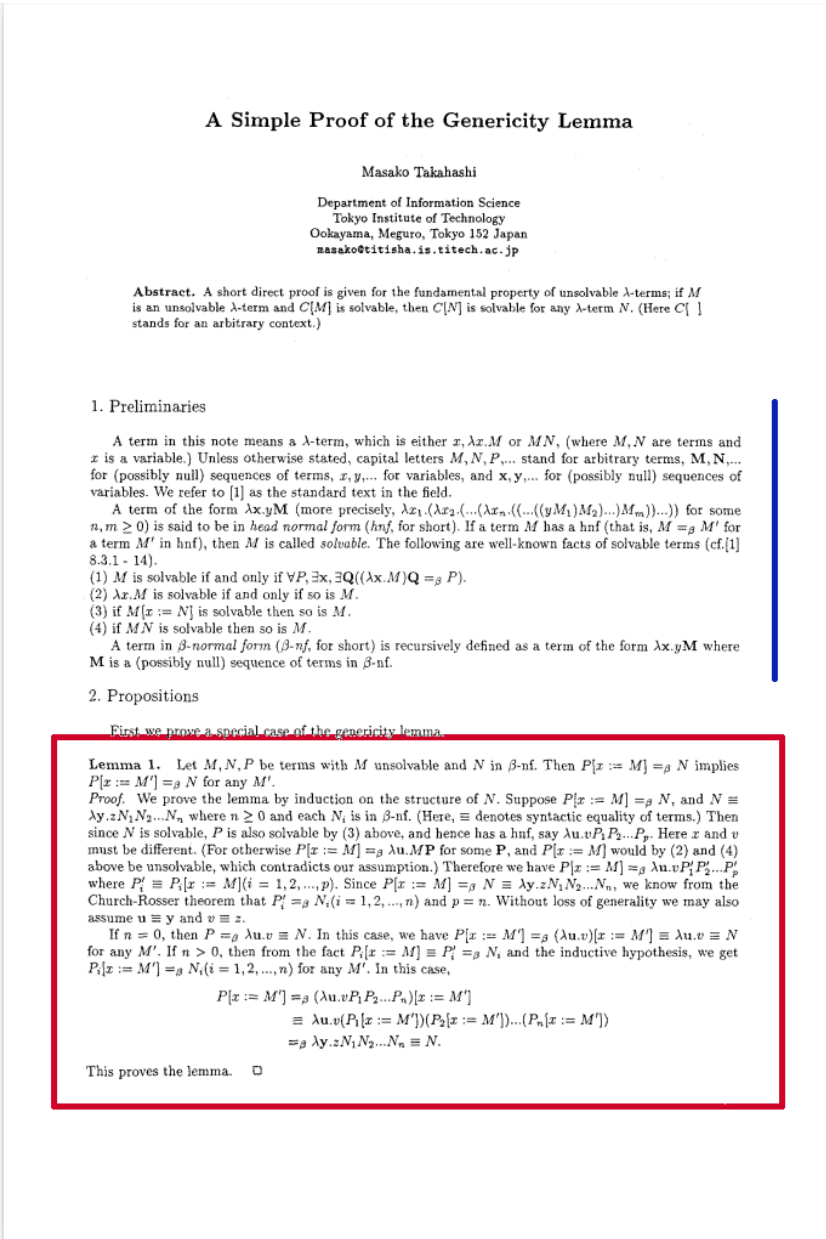
Main Lemma

~15 lines of text

~90 lines of Abella

~140 tactics

Proving/Formalizing Genericity



A Simple Proof of the Genericity Lemma

Masako Takahashi

Department of Information Science
Tokyo Institute of Technology
Ookayama, Meguro, Tokyo 152 Japan
masako@titisha.is.titech.ac.jp

Abstract. A short direct proof is given for the fundamental property of unsolvable λ-terms; if M is an unsolvable λ-term and $C[M]$ is solvable, then $C[N]$ is solvable for any λ-term N. (Here $C[\ ]$ stands for an arbitrary context.)

1. Preliminaries

A term in this note means a λ-term, which is either $x, \lambda x.M$ or MN, (where M, N are terms and x is a variable.) Unless otherwise stated, capital letters $M, N, P, ...$ stand for arbitrary terms, $\mathbf{M}, \mathbf{N}, ...$ for (possibly null) sequences of terms, $x, y, ...$ for variables, and $\mathbf{x}, \mathbf{y}, ...$ for (possibly null) sequences of variables. We refer to [1] as the standard text in the field.

A term of the form $\lambda \mathbf{x}.y\mathbf{M}$ (more precisely, $\lambda x_1.(\lambda x_2.(...(\lambda x_n.((...((yM_1)M_2)...)M_m))...))$ for some $n, m \geq 0$) is said to be in *head normal form* (*hnf*, for short). If a term M has a hnf (that is, $M =_\beta M'$ for a term M' in hnf), then M is called *solvable*. The following are well-known facts of solvable terms (cf.[1] 8.3.1 - 14).

(1) M is solvable if and only if $\forall P, \exists \mathbf{x}, \exists \mathbf{Q}((\lambda x.M)\mathbf{Q} =_\beta P)$.
(2) $\lambda x.M$ is solvable if and only if so is M.
(3) if $M[x := N]$ is solvable then so is M.
(4) if MN is solvable then so is M.

A term in *β-normal form* (*β-nf*, for short) is recursively defined as a term of the form $\lambda \mathbf{x}.y\mathbf{M}$ where $\mathbf{M}$ is a (possibly null) sequence of terms in β-nf.

2. Propositions

First we prove a special case of the genericity lemma.

Lemma 1. Let M, N, P be terms with M unsolvable and N in β-nf. Then $P[x := M] =_\beta N$ implies $P[x := M'] =_\beta N$ for any M'.

Proof. We prove the lemma by induction on the structure of N. Suppose $P[x := M] =_\beta N$, and $N \equiv \lambda y.zN_1N_2...N_n$ where $n \geq 0$ and each N_i is in β-nf. (Here, $\equiv$ denotes syntactic equality of terms.) Then since N is solvable, P is also solvable by (3) above, and hence has a hnf, say $\lambda u.vP_1P_2...P_p$. Here x and v must be different. (For otherwise $P[x := M] =_\beta \lambda u.M\mathbf{P}$ for some $\mathbf{P}$, and $P[x := M]$ would by (2) and (4) above be unsolvable, which contradicts our assumption.) Therefore we have $P[x := M] =_\beta \lambda u.vP'_1P'_2...P'_p$ where $P'_i \equiv P_i[x := M] (i = 1, 2, ..., p)$. Since $P[x := M] =_\beta N \equiv \lambda y.zN_1N_2...N_n$, we know from the Church-Rosser theorem that $P'_i =_\beta N_i (i = 1, 2, ..., n)$ and $p = n$. Without loss of generality we may also assume $\mathbf{u} \equiv \mathbf{y}$ and $v \equiv z$.

If $n = 0$, then $P =_\beta \lambda u.v \equiv N$. In this case, we have $P[x := M'] =_\beta (\lambda u.v)[x := M'] \equiv \lambda u.v \equiv N$ for any M'. If $n > 0$, then from the fact $P_i[x := M] \equiv P'_i =_\beta N_i$ and the inductive hypothesis, we get $P_i[x := M'] =_\beta N_i (i = 1, 2, ..., n)$ for any M'. In this case,

$$\begin{aligned} P[x := M'] &=_\beta (\lambda u.vP_1P_2...P_n)[x := M'] \\ &\equiv \lambda u.v(P_1[x := M'])(P_2[x := M'])...(P_n[x := M']) \\ &=_\beta \lambda y.zN_1N_2...N_n \equiv N. \end{aligned}$$

This proves the lemma. □

Lemma 2. ([1] 14.3.24. Genericity lemma) Let M be an unsolvable term, and $C[\ ]$ be a context such that $C[M]$ has a β-nf. Then $C[M] =_\beta C[M']$ for any M'.

Proof. For given M', let $\mathbf{y}$ be a sequence of all free variables in MM'. Take a new variable x (neither in $C[M]$ nor $C[M']$), and let $P \equiv C[x\mathbf{y}]$. Then since $\lambda \mathbf{y}.M$ and $\lambda \mathbf{y}.M'$ are closed terms, we have

$$P[x := \lambda \mathbf{y}.M] \equiv C[(\lambda \mathbf{y}.M)\mathbf{y}] =_\beta C[M],$$
$$P[x := \lambda \mathbf{y}.M'] \equiv C[(\lambda \mathbf{y}.M')\mathbf{y}] =_\beta C[M'].$$

The term $\lambda \mathbf{y}.M$ therefore satisfies $P[x := \lambda \mathbf{y}.M] =_\beta C[M] =_\beta N$ for some N in β-nf. Here $\lambda \mathbf{y}.M$ is unsolvable because so is M. Hence by applying lemma 1 we get $P[x := \lambda \mathbf{y}.M'] =_\beta N$, which implies $C[M] =_\beta C[M']$. □

Corollary 3. If M is unsolvable and $C[M]$ is solvable, then $C[M']$ is solvable for any M'.

Proof. Since $C[M]$ is solvable, by (1) above there exist $\mathbf{x}$ and $\mathbf{N}$ such that $(\lambda x.C[M])\mathbf{N}$ has a β-nf. Then by lemma 2 (applied to the context $(\lambda x.C[\ ])\mathbf{N}$), we know $(\lambda x.C[M'])\mathbf{N}$ has a β-nf for any M'. This means $(\lambda x.C[M'])\mathbf{N}$ is solvable, and consequently $C[M']$ is solvable. □

The proof presented provides an alternative to the conventional one which uses a topological argument on Böhm trees (cf.[1] Chapters 10 and 14).

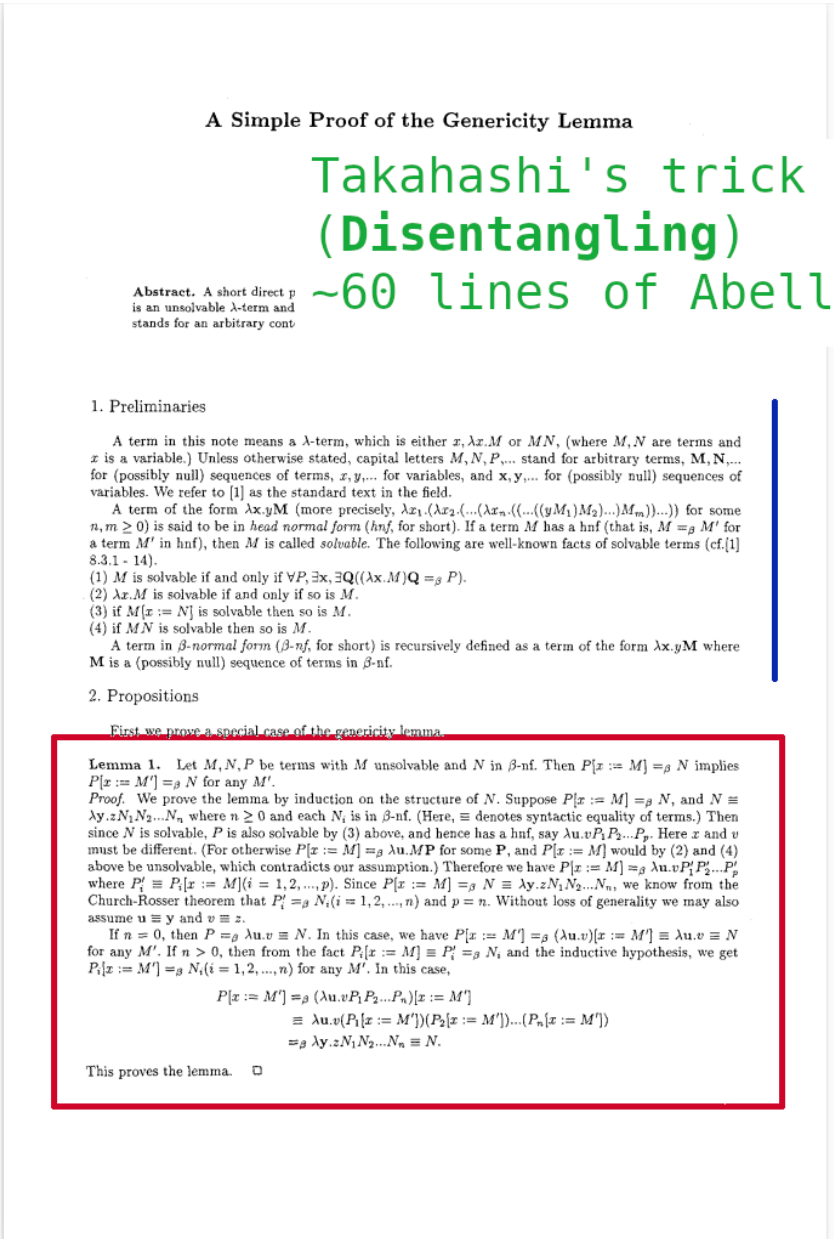
Reference

[1] H. P. Barendregt, *The Lambda Calculus* (North-Holland 1984).

Preliminaries:
~2000 lines of Abella

Main Lemma
~15 lines of text
~90 lines of Abella
~140 tactics

Proving/Formalizing Genericity



A Simple Proof of the Genericity Lemma

Takahashi's trick
(**Disentangling**)
~60 lines of Abella

Abstract. A short direct p
is an unsolvable λ-term and
stands for an arbitrary cont

1. Preliminaries

A term in this note means a λ-term, which is either $x, \lambda x.M$ or MN, (where M, N are terms and x is a variable.) Unless otherwise stated, capital letters $M, N, P, ...$ stand for arbitrary terms, $\mathbf{M}, \mathbf{N}, ...$ for (possibly null) sequences of terms, $x, y, ...$ for variables, and $\mathbf{x}, \mathbf{y}, ...$ for (possibly null) sequences of variables. We refer to [1] as the standard text in the field.

A term of the form $\lambda \mathbf{x}.y\mathbf{M}$ (more precisely, $\lambda x_1.(\lambda x_2.(...(\lambda x_n.((...((yM_1)M_2)...)M_m))...))$ for some $n, m \geq 0$) is said to be in *head normal form* (*hnf*, for short). If a term M has a hnf (that is, $M =_\beta M'$ for a term M' in hnf), then M is called *solvable*. The following are well-known facts of solvable terms (cf.[1] 8.3.1 - 14).

(1) M is solvable if and only if $\forall P, \exists \mathbf{x}, \exists \mathbf{Q}((\lambda x.M)\mathbf{Q} =_\beta P)$.
(2) $\lambda x.M$ is solvable if and only if so is M.
(3) if $M[x := N]$ is solvable then so is M.
(4) if MN is solvable then so is M.

A term in *β-normal form* (*β-nf*, for short) is recursively defined as a term of the form $\lambda \mathbf{x}.y\mathbf{M}$ where $\mathbf{M}$ is a (possibly null) sequence of terms in β-nf.

2. Propositions

First we prove a special case of the genericity lemma.

Lemma 1. Let M, N, P be terms with M unsolvable and N in β-nf. Then $P[x := M] =_\beta N$ implies $P[x := M'] =_\beta N$ for any M'.

Proof. We prove the lemma by induction on the structure of N. Suppose $P[x := M] =_\beta N$, and $N \equiv \lambda y.zN_1N_2...N_n$ where $n \geq 0$ and each N_i is in β-nf. (Here, $\equiv$ denotes syntactic equality of terms.) Then since N is solvable, P is also solvable by (3) above, and hence has a hnf, say $\lambda u.vP_1P_2...P_p$. Here x and v must be different. (For otherwise $P[x := M] =_\beta \lambda u.M\mathbf{P}$ for some $\mathbf{P}$, and $P[x := M]$ would by (2) and (4) above be unsolvable, which contradicts our assumption.) Therefore we have $P[x := M] =_\beta \lambda u.vP'_1P'_2...P'_p$ where $P'_i \equiv P_i[x := M] (i = 1, 2, ..., p)$. Since $P[x := M] =_\beta N \equiv \lambda y.zN_1N_2...N_n$, we know from the Church-Rosser theorem that $P'_i =_\beta N_i (i = 1, 2, ..., n)$ and $p = n$. Without loss of generality we may also assume $\mathbf{u} \equiv \mathbf{y}$ and $v \equiv z$.

If $n = 0$, then $P =_\beta \lambda u.v \equiv N$. In this case, we have $P[x := M'] =_\beta (\lambda u.v)[x := M'] \equiv \lambda u.v \equiv N$ for any M'. If $n > 0$, then from the fact $P_i[x := M] \equiv P'_i =_\beta N_i$ and the inductive hypothesis, we get $P_i[x := M'] =_\beta N_i (i = 1, 2, ..., n)$ for any M'. In this case,

$$\begin{aligned} P[x := M'] &=_\beta (\lambda u.vP_1P_2...P_n)[x := M'] \\ &\equiv \lambda u.v(P_1[x := M'])(P_2[x := M'])...(P_n[x := M']) \\ &=_\beta \lambda y.zN_1N_2...N_n \equiv N. \end{aligned}$$

This proves the lemma. □

118

Lemma 2. ([1] 14.3.24. Genericity lemma) Let M be an unsolvable term, and $C[\ ]$ be a context such that $C[M]$ has a β-nf. Then $C[M] =_\beta C[M']$ for any M'.

Proof. For given M', let $\mathbf{y}$ be a sequence of all free variables in MM'. Take a new variable x (neither in $C[M]$ nor $C[M']$), and let $P \equiv C[x\mathbf{y}]$. Then since $\lambda y.M$ and $\lambda y.M'$ are closed terms, we have

$$P[x := \lambda y.M] \equiv C[(\lambda y.M)\mathbf{y}] =_\beta C[M],$$
$$P[x := \lambda y.M'] \equiv C[(\lambda y.M')\mathbf{y}] =_\beta C[M'].$$

The term $\lambda y.M$ therefore satisfies $P[x := \lambda y.M] =_\beta C[M] =_\beta N$ for some N in β-nf. Here $\lambda y.M$ is unsolvable because so is M. Hence by applying lemma 1 we get $P[x := \lambda y.M'] =_\beta N$, which implies $C[M] =_\beta C[M']$. □

Corollary 3. If M is unsolvable and $C[M]$ is solvable, then $C[M']$ is solvable for any M'.

Proof. Since $C[M]$ is solvable, by (1) above there exist $\mathbf{x}$ and $\mathbf{N}$ such that $(\lambda x.C[M])\mathbf{N}$ has a β-nf. Then by lemma 2 (applied to the context $(\lambda x.C[\ ])\mathbf{N}$), we know $(\lambda x.C[M'])\mathbf{N}$ has a β-nf for any M'. This means $(\lambda x.C[M'])\mathbf{N}$ is solvable, and consequently $C[M']$ is solvable. □

The proof presented provides an alternative to the conventional one which uses a topological argument on Böhm trees (cf.[1] Chapters 10 and 14).

Reference

[1] H. P. Barendregt, *The Lambda Calculus* (North-Holland 1984).

Preliminaries:
~2000 lines of Abella

Main Lemma
~15 lines of text
~90 lines of Abella
~140 tactics

This Talk

- Why is (light) genericity useful?
- Formalizing the theory of the λ-calculus in Abella
- Formalizing Takahashi's proof of genericity

(In)Equational Theories

Fix an evaluation strategy $\mathtt{s}$ within a language of terms $(\mathcal{L}, \rightarrow_{\mathcal{L}})$, $\rightarrow_{\mathtt{s}} \subseteq \rightarrow_{\mathcal{L}}$.

An inequational theory $=_{\mathcal{T}}$ for $\mathtt{s}$ is:

1. an *Equivalence Relation* on terms, $=_{\mathcal{T}} \subseteq \mathcal{L} \times \mathcal{L}$;
2. *Context-Closed*:
$$\text{if } t =_{\mathcal{T}} u \text{ then } \forall C,\ C\langle t \rangle =_{\mathcal{T}} C\langle u \rangle;$$
3. *Invariant under Computation*:
$$\text{if } t =_{\mathcal{L}} u \text{ then } t =_{\mathcal{T}} u.$$

(In)Equational Theories

Fix an evaluation strategy $\mathtt{s}$ within a language of terms $(\mathcal{L}, \rightarrow_{\mathcal{L}})$, $\rightarrow_{\mathtt{s}} \subseteq \rightarrow_{\mathcal{L}}$.

An inequational theory $=_{\mathcal{T}}$ for $\mathtt{s}$ is:

1. an *Equivalence Relation* on terms, $=_{\mathcal{T}} \subseteq \mathcal{L} \times \mathcal{L}$;
2. *Context-Closed*:
$$\text{if } t =_{\mathcal{T}} u \text{ then } \forall C,\ C\langle t \rangle =_{\mathcal{T}} C\langle u \rangle;$$
3. *Invariant under Computation*:
$$\text{if } t =_{\mathcal{L}} u \text{ then } t =_{\mathcal{T}} u.$$

(In)Equational Theories

Fix an evaluation strategy $\mathtt{s}$ within a language of terms $(\mathcal{L}, \to_{\mathcal{L}})$, $\to_{\mathtt{s}} \subseteq \to_{\mathcal{L}}$.

An inequational theory $\leq_{\mathcal{T}}$ for $\mathtt{s}$ is:

1. a Preorder on terms, $\leq_{\mathcal{T}} \subseteq \mathcal{L} \times \mathcal{L}$;
2. *Context-Closed*:

$$\text{if } t \leq_{\mathcal{T}} u \text{ then } \forall C,\ C\langle t \rangle \leq_{\mathcal{T}} C\langle u \rangle;$$

3. *Invariant under Computation*:

$$\text{if } t =_{\mathcal{L}} u \text{ then } t \leq_{\mathcal{T}} u.$$

(In)Equational Theories

Fix an evaluation strategy $\mathtt{s}$ within a language of terms $(\mathcal{L}, \rightarrow_{\mathcal{L}})$, $\rightarrow_{\mathtt{s}} \subseteq \rightarrow_{\mathcal{L}}$.

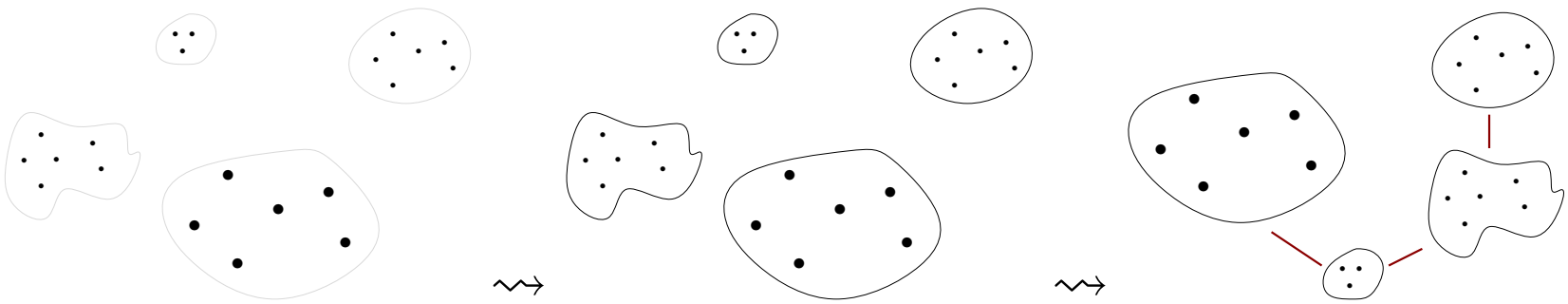
An inequational theory $\leq_{\mathcal{T}}$ for $\mathtt{s}$ is:

1. a Preorder on terms, $\leq_{\mathcal{T}} \subseteq \mathcal{L} \times \mathcal{L}$;
2. *Context-Closed*:

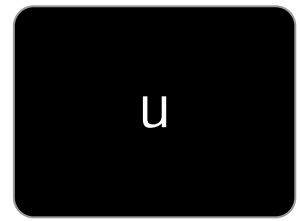
$$\text{if } t \leq_{\mathcal{T}} u \text{ then } \forall C,\ C\langle t \rangle \leq_{\mathcal{T}} C\langle u \rangle;$$

3. *Invariant under Computation*:

$$\text{if } t =_{\mathcal{L}} u \text{ then } t \leq_{\mathcal{T}} u.$$



Contextual Equivalence and Preorder



When are two λ-terms equivalent? [Mor68]

$t \equiv_{\mathsf{s}}^{\mathrm{octx}} u$ if for all contexts C . $[\ C\langle t\rangle \Downarrow_{\mathsf{s}} \ \Leftrightarrow \ C\langle u\rangle \Downarrow_{\mathsf{s}}\ ]$

$t \sqsubseteq_{\mathsf{s}}^{\mathrm{octx}} u$ if for all contexts C . $[\ C\langle t\rangle \Downarrow_{\mathsf{s}} \ \Rightarrow \ C\langle u\rangle \Downarrow_{\mathsf{s}}\ ]$

The s-contextual preorder is generally an inequational theory.

Contextual Equivalence and Preorder

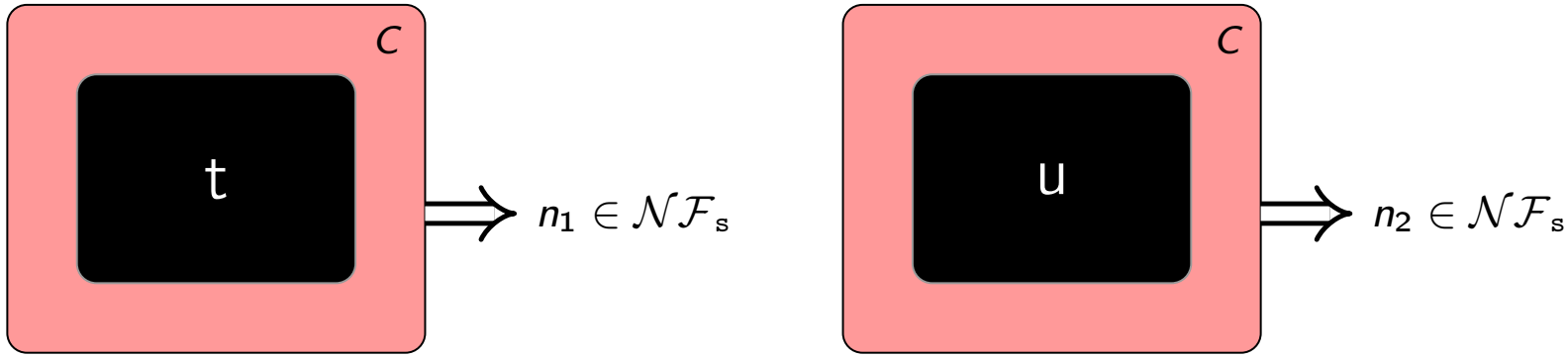
When are two λ-terms equivalent? [Mor68]

$$t \equiv_{\mathbf{s}}^{\mathrm{octx}} u \quad \text{if for all contexts } C\,. \quad [\, C\langle t\rangle \Downarrow_{\mathbf{s}} \quad \Leftrightarrow \quad C\langle u\rangle \Downarrow_{\mathbf{s}} \,]$$

$$t \sqsubseteq_{\mathbf{s}}^{\mathrm{octx}} u \quad \text{if for all contexts } C\,. \quad [\, C\langle t\rangle \Downarrow_{\mathbf{s}} \quad \Rightarrow \quad C\langle u\rangle \Downarrow_{\mathbf{s}} \,]$$

The s-contextual preorder is generally an inequational theory.

Contextual Equivalence and Preorder



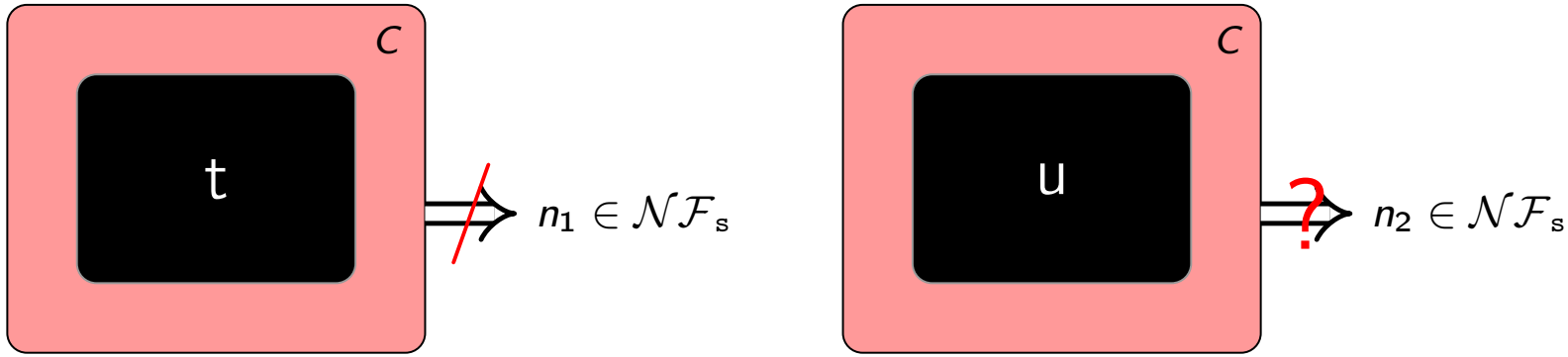
When are two λ-terms equivalent? [Mor68]

$$t \equiv_s^{\mathrm{octx}} u \quad \text{if for all contexts } C\,. \quad [\, C\langle t\rangle \Downarrow_s \quad \Leftrightarrow \quad C\langle u\rangle \Downarrow_s \,]$$

$$t \sqsubseteq_s^{\mathrm{octx}} u \quad \text{if for all contexts } C\,. \quad [\, C\langle t\rangle \Downarrow_s \quad \Rightarrow \quad C\langle u\rangle \Downarrow_s \,]$$

The s-contextual preorder is generally an inequational theory.

Contextual Equivalence and Preorder



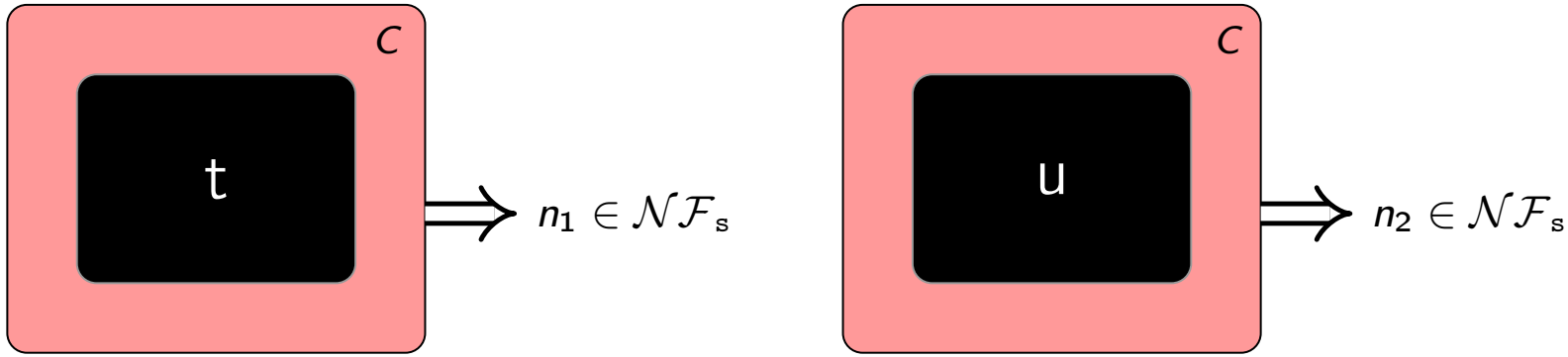
When are two λ-terms equivalent? [Mor68]

$$t \equiv_{\mathbf{s}}^{\mathrm{octx}} u \quad \text{if for all contexts } C\,. \quad [\; C\langle t\rangle \Downarrow_{\mathbf{s}} \quad \Leftrightarrow \quad C\langle u\rangle \Downarrow_{\mathbf{s}} \;]$$

$$t \sqsubseteq_{\mathbf{s}}^{\mathrm{octx}} u \quad \text{if for all contexts } C\,. \quad [\; C\langle t\rangle \Downarrow_{\mathbf{s}} \quad \Rightarrow \quad C\langle u\rangle \Downarrow_{\mathbf{s}} \;]$$

The s-contextual preorder is generally an inequational theory.

Contextual Equivalence and Preorder



When are two λ-terms equivalent? [Mor68]

$$t \equiv_{\mathbf{s}}^{\mathrm{octx}} u \quad \text{if for all contexts } C\,. \quad [\; C\langle t\rangle \Downarrow_{\mathbf{s}} \quad \Leftrightarrow \quad C\langle u\rangle \Downarrow_{\mathbf{s}} \;]$$

$$t \sqsubseteq_{\mathbf{s}}^{\mathrm{octx}} u \quad \text{if for all contexts } C\,. \quad [\; C\langle t\rangle \Downarrow_{\mathbf{s}} \quad \Rightarrow \quad C\langle u\rangle \Downarrow_{\mathbf{s}} \;]$$

The s-contextual preorder is generally an inequational theory.

Why Contextual Equivalence?

The contextual preorder is a natural extensional principle, but that can be quite hard to use.

« The contextual preorder is the largest sensible inequational theory to study. »

Maximal Theory:

A theory $\leq_{\mathcal{T}}$ is maximal if for any theory $\leq_{\mathcal{T}'}$ we have that if $\leq_{\mathcal{T}} \subsetneq \leq_{\mathcal{T}'}$ then $\leq_{\mathcal{T}'}$ is inconsistent.

Why Contextual Equivalence?

The contextual preorder is a **natural** extensional principle, but that can be quite hard to use.

« The contextual preorder is the largest sensible inequational theory to study. »

Maximal Theory:

A theory $\leq_{\mathcal{T}}$ is maximal if for any theory $\leq_{\mathcal{T}'}$ we have that if $\leq_{\mathcal{T}} \subsetneq \leq_{\mathcal{T}'}$ then $\leq_{\mathcal{T}'}$ is inconsistent.

Why Contextual Equivalence?

The contextual preorder is a **natural** extensional principle, but that can be quite hard to use.

« The contextual preorder is the largest sensible inequational theory to study. »

Maximal Theory:

A theory $\leq_{\mathcal{T}}$ is maximal if for any theory $\leq_{\mathcal{T}'}$ we have that if $\leq_{\mathcal{T}} \subsetneq \leq_{\mathcal{T}'}$ then $\leq_{\mathcal{T}'}$ is inconsistent.

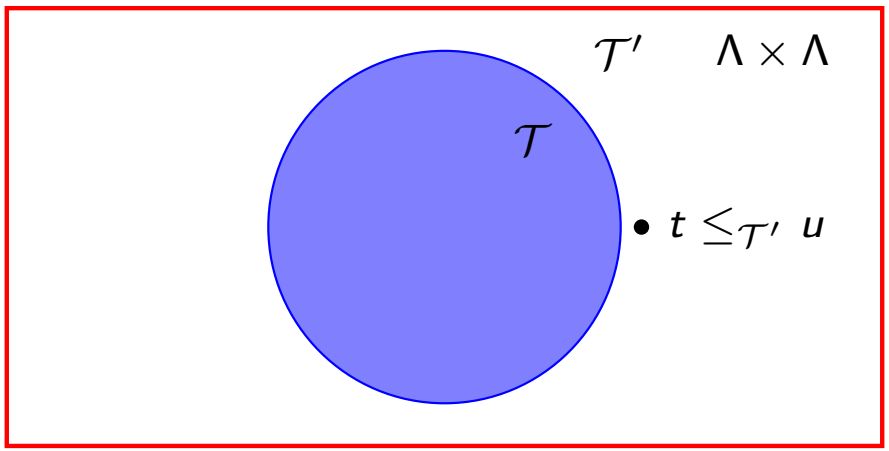
Why Contextual Equivalence?

The contextual preorder is a **natural** extensional principle, but that can be quite hard to use.

« The contextual preorder is the largest sensible inequational theory to study. »

Maximal Theory:

A theory $\leq_{\mathcal{T}}$ is maximal if for any theory $\leq_{\mathcal{T}'}$ we have that if $\leq_{\mathcal{T}} \subsetneq \leq_{\mathcal{T}'}$ then $\leq_{\mathcal{T}'}$ is inconsistent.



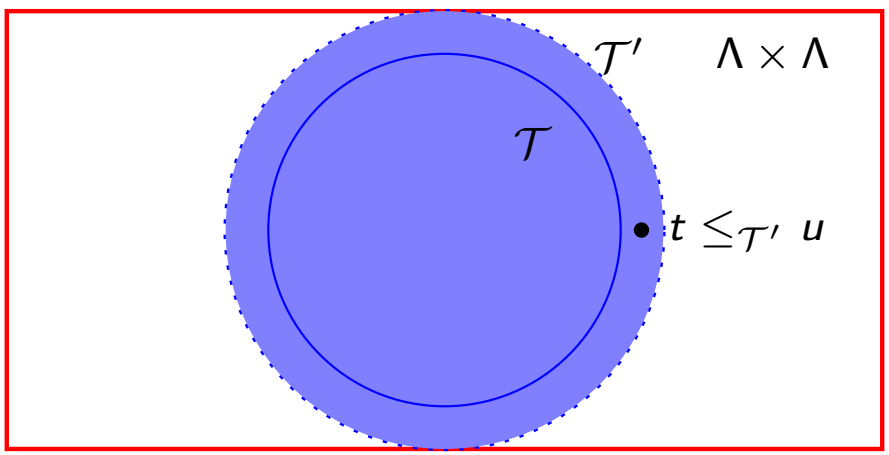
Why Contextual Equivalence?

The contextual preorder is a **natural** extensional principle, but that can be quite hard to use.

« The contextual preorder is the largest sensible inequational theory to study. »

Maximal Theory:

A theory $\leq_{\mathcal{T}}$ is maximal if for any theory $\leq_{\mathcal{T}'}$ we have that if $\leq_{\mathcal{T}} \subsetneq \leq_{\mathcal{T}'}$ then $\leq_{\mathcal{T}'}$ is inconsistent.



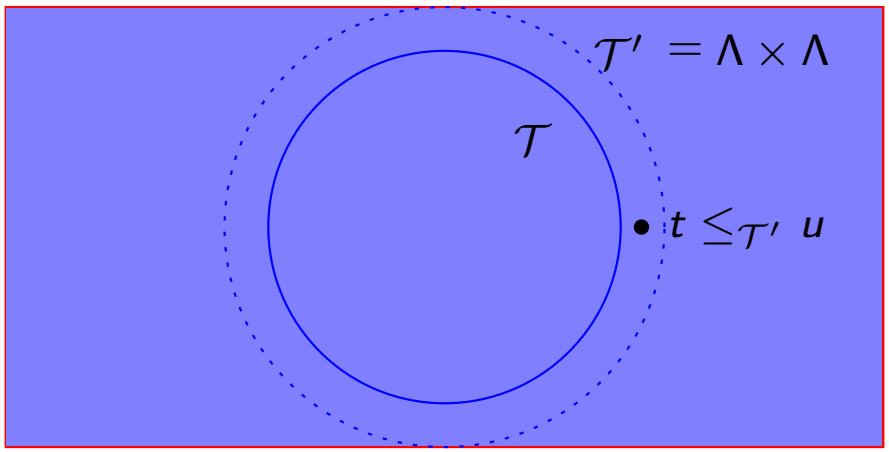
Why Contextual Equivalence?

The contextual preorder is a **natural** extensional principle, but that can be quite hard to use.

« The contextual preorder is the largest sensible inequational theory to study. »

Maximal Theory:

A theory $\leq_{\mathcal{T}}$ is maximal if for any theory $\leq_{\mathcal{T}'}$ we have that if $\leq_{\mathcal{T}} \subsetneq \leq_{\mathcal{T}'}$ then $\leq_{\mathcal{T}'}$ is inconsistent.



Sufficient Conditions for Maximality

Theorem (**Maximality of Contextual Preorders**)

For s *a* ***dissecting*** *strategy which enjoys light genericity, the contextual preorder* $\sqsubseteq_{=\mathbf{s}}^{\mathrm{octx}}$ *is maximal.*

Factorizing: Proofs $\rightarrow$ 1 parametric proof via **dissecting** strategies:

for every $t \Downarrow_{\mathsf{s}}$ and $u \not\Downarrow_{\mathsf{s}}$, for every s there exists C such that $C\langle t\rangle =_{\mathsf{s}} s$ and $C\langle u\rangle \not\Downarrow_{\mathsf{s}}$.

Simplifying: CbN proof relied on Böhm's theorem $\rightarrow$ proof relying on **light genericity**:

LG: s-diverging terms are minimum elements for the contextual preorder.

Sufficient Conditions for Maximality

Theorem (**Maximality of Contextual Preorders**)

For $\mathtt{s}$ *a* ***dissecting*** *strategy which enjoys light genericity, the contextual preorder* $\sqsubseteq_{=\mathbf{s}}^{\mathrm{octx}}$ *is maximal.*

Factorizing: Proofs $\rightarrow$ 1 parametric proof via **dissecting** strategies:

for every $t \Downarrow_{\mathtt{s}}$ and $u \not\Downarrow_{\mathtt{s}}$, for every s there exists C such that $C\langle t\rangle =_{\mathtt{s}} s$ and $C\langle u\rangle \not\Downarrow_{\mathtt{s}}$.

Simplifying: CbN proof relied on Böhm's theorem $\rightarrow$ proof relying on **light genericity**:

LG: s-diverging terms are minimum elements for the contextual preorder.

Sufficient Conditions for Maximality

Theorem (**Maximality of Contextual Preorders**)

For $\mathtt{s}$ *a* ***dissecting*** *strategy which enjoys light genericity, the contextual preorder* $\sqsubseteq_{=\mathtt{s}}^{\mathrm{octx}}$ *is maximal.*

Factorizing: Proofs $\rightarrow$ 1 parametric proof via **dissecting** strategies:

for every $t \Downarrow_{\mathtt{s}}$ and $u \not\Downarrow_{\mathtt{s}}$, for every s there exists C such that $C\langle t\rangle =_{\mathtt{s}} s$ and $C\langle u\rangle \not\Downarrow_{\mathtt{s}}$.

Simplifying: CbN proof relied on Böhm's theorem $\rightarrow$ proof relying on light genericity:

LG: s-diverging terms are minimum elements for the contextual preorder.

Sufficient Conditions for Maximality

Theorem (**Maximality of Contextual Preorders**)

For s *a* ***dissecting*** *strategy which enjoys light genericity, the contextual preorder* $\sqsubseteq_{=\mathbf{s}}^{\mathrm{octx}}$ *is maximal.*

Factorizing: Proof~~s~~ $\rightarrow$ 1 parametric proof via **dissecting** strategies:

for every $t \Downarrow_{\mathbf{s}}$ and $u \not\Downarrow_{\mathbf{s}}$, for every s there exists C such that $C\langle t\rangle =_{\mathbf{s}} s$ and $C\langle u\rangle \not\Downarrow_{\mathbf{s}}$.

Simplifying: CbN proof relied on Böhm's theorem $\rightarrow$ proof relying on **light genericity**:

LG: s-diverging terms are minimum elements for the contextual preorder.

Sufficient Conditions for Maximality

Theorem (**Maximality of Contextual Preorders**)

For s *a* ***dissecting*** *strategy which enjoys light genericity, the contextual preorder* $\sqsubseteq_{=\mathtt{s}}^{\mathrm{octx}}$ *is maximal.*

Factorizing: Proof~~s~~ $\rightarrow$ 1 parametric proof via **dissecting** strategies:

for every $t \Downarrow_{\mathtt{s}}$ and $u \not\Downarrow_{\mathtt{s}}$, for every s there exists C such that $C\langle t\rangle =_{\mathtt{s}} s$ and $C\langle u\rangle \not\Downarrow_{\mathtt{s}}$.

Simplifying: CbN proof relied on ~~Böhm's theorem~~ $\rightarrow$ proof relying on **light genericity**:

LG: s-diverging terms are minimum elements for the contextual preorder.

Light Genericity in Call-by-Name

Light Genericity in Call-by-Name is stated using head reduction.

CbN Light Genericity: head-diverging terms are minimum for the head open contextual preorder.

Which unfolds to:

CbN Light Genericity: let u be head-diverging and C such that $C\langle u\rangle$ is head-normalizing then $C\langle t\rangle$ is head-normalizing for all $t \in \Lambda$.

Main difficulty: reasoning with contexts and reduction.

Light Genericity in Call-by-Name

Light Genericity in Call-by-Name is stated using head reduction.

CbN Light Genericity: head-diverging terms are minimum for the head open contextual preorder.

Which unfolds to:

CbN Light Genericity: let u be head-diverging and C such that $C\langle u\rangle$ is head-normalizing then $C\langle t\rangle$ is head-normalizing for all $t \in \Lambda$.

Main difficulty: reasoning with contexts and reduction.

Proofs of Call-by-Name (Light) Genericity

- Original Topological proof by Barendregt
- Operational Proof by Takahashi
- Taylor Expansion proof of Barbarossa and Manzonetto
- Approximants Calculus by Arrial, Guerrieri and Kesner
- Proofs using Semantic Models
- Many Others

Light Genericity in Call-by-Name

Takahashi proves heavy genericity with a very short proof [Tak94] and gives as a corollary light genericity.

Key idea: Reason with substitutions instead of contexts!

In the paper, we adapt **Takahashi's technique** to give a direct proof of light genericity.

Light genericity as substitution: let u be s-diverging and t such that $t\{x \leftarrow u\}$ is s-normalizing then $t\{x \leftarrow s\}$ is s-normalizing for all $s \in \Lambda$.

Light Genericity in Call-by-Name

Takahashi proves heavy genericity with a very short proof [Tak94] and gives as a corollary light genericity.

Key idea: Reason with substitutions instead of contexts!

In the paper, we adapt **Takahashi's technique** to give a direct proof of light genericity.

Light genericity as substitution: let u be s-diverging and t such that $t\{x \leftarrow u\}$ is s-normalizing then $t\{x \leftarrow s\}$ is s-normalizing for all $s \in \Lambda$.

Formalizing λ

$$\text{Terms} \quad t, u \quad := \quad x \mid \lambda x.t \mid tu$$

$$\frac{}{(\lambda x.t)u \mapsto_\beta t\{x \leftarrow u\}} \qquad \frac{t \to_\beta t'}{tu \to_\beta t'u} \qquad \frac{t \to_\beta t'}{\lambda x.t \to_\beta \lambda x.t'} \qquad \frac{u \to_\beta u'}{tu \to_\beta tu'}$$

$$\frac{}{(\lambda x.t)u \to_{\mathtt{h}} t\{x \leftarrow u\}} \qquad \frac{t \to_{\mathtt{h}} u}{ts \to_{\mathtt{h}} us} \qquad \frac{t \to_{\mathtt{h}} u}{\lambda x.t \to_{\mathtt{h}} \lambda x.u}$$

Formalizing λ-theories

Definition (Inequational theory)

An inequational theory $\leq_{\mathcal{T}}$ is a pre-order on terms such that:

- *Stable by contextual closure*:
  $t \leq_{\mathcal{T}} u \implies \forall C,\ C\langle t\rangle \leq_{\mathcal{T}} C\langle u\rangle$;
- *Computational Invariance*: it contains β-conversion.

Head (open) contextual preorders is an inequational theory.

The non-trivial point is that it contains β-conversion.

Formalizing λ-theories

Definition (Inequational theory)

An inequational theory $\leq_{\mathcal{T}}$ is a pre-order on terms such that:

- *Stable by contextual closure*:
  $t \leq_{\mathcal{T}} u \implies \forall C,\ C\langle t\rangle \leq_{\mathcal{T}} C\langle u\rangle$;
- *Computational Invariance*: it contains β-conversion.

Head (open) contextual preorders is an inequational theory.

The non-trivial point is that it contains β-conversion.

Rewriting results: Normalization

Theorem (Head Normalization)

Let t be a term. If $t \to_\beta^* u$ and u does not $\to_{\mathtt{h}}$-reduce then $\to_{\mathtt{h}}$ terminates on t.

Proven by factorization/standardization.
(We also formalize confluence but come one everyone has done it once in their life!)

Rewriting results: Normalization

Theorem (Head Normalization)

Let t be a term. If $t \to_\beta^* u$ and u does not $\to_{\mathtt{h}}$-reduce then $\to_{\mathtt{h}}$ terminates on t.

Proven by factorization/standardization.
(We also formalize confluence but come one everyone has done it once in their life!)

Solvability Characterization

The following are equivalent, given a term t:

- *Solvability*: there exists a head context H such that $H\langle t\rangle \to_\beta^* \mathtt{I} := \lambda x.x$;
- *Head Normalization*: t is head normalizing.

Proven via the head normalization theorem.

Formalizing Contexts

A context is a term with a hole? Not really...

For $C := \lambda x.\langle\cdot\rangle$, we have that $C\langle x\rangle = \lambda x.x$ and $C\langle y\rangle = \lambda x.y$

A context is a term with a hole *annotated by the possible captured variables*...

$$\lambda x.\Box_x$$

A context is a function from `tm` to `tm`, coded in Abella as a predicate:

`ctx T CT` holds iff there exists a context C such that $C\langle \texttt{T}\rangle = \texttt{CT}$.

Applying a context to two different terms:

`ctxs T CT U CU` holds iff there exists a context C such that $C\langle \texttt{T}\rangle = \texttt{CT}$ and $C\langle \texttt{U}\rangle = \texttt{CU}$.

Formalizing Contexts

A context is a term with a hole? Not really...

For $C := \lambda x.\langle\cdot\rangle$, we have that $C\langle x\rangle = \lambda x.x$ and $C\langle y\rangle = \lambda x.y$

A context is a term with a hole *annotated by the possible captured variables*...

$$\lambda x.\square_x$$

A context is a function from `tm` to `tm`, coded in Abella as a predicate:

`ctx T CT` holds iff there exists a context C such that $C\langle \texttt{T}\rangle = \texttt{CT}$.

Applying a context to two different terms:

`ctxs T CT U CU` holds iff there exists a context C such that $C\langle \texttt{T}\rangle = \texttt{CT}$ and $C\langle \texttt{U}\rangle = \texttt{CU}$.

Formalizing Contexts

A context is a term with a hole? Not really...

For $C := \lambda x.\langle\cdot\rangle$, we have that $C\langle x\rangle = \lambda x.x$ and $C\langle y\rangle = \lambda x.y$

A context is a term with a hole *annotated by the possible captured variables*...

$$\lambda x.\square_x$$

A context is a function from `tm` to `tm`, coded in Abella as a predicate:

`ctx T CT` holds iff there exists a context C such that $C\langle \texttt{T}\rangle = \texttt{CT}$.

Applying a context to two different terms:

`ctxs T CT U CU` holds iff there exists a context C such that $C\langle \texttt{T}\rangle = \texttt{CT}$ and $C\langle \texttt{U}\rangle = \texttt{CU}$.

Formalizing Contexts

A context is a term with a hole? Not really...

For $C := \lambda x.\langle\cdot\rangle$, we have that $C\langle x\rangle = \lambda x.x$ and $C\langle y\rangle = \lambda x.y$

A context is a term with a hole *annotated by the possible captured variables*...

$$\lambda x.\Box_x$$

A context is a function from `tm` to `tm`, coded in Abella as a predicate:

`ctx T CT` holds iff there exists a context C such that $C\langle \texttt{T}\rangle = \texttt{CT}$.

Applying a context to two different terms:

`ctxs T CT U CU` holds iff there exists a context C such that $C\langle \texttt{T}\rangle = \texttt{CT}$ and $C\langle \texttt{U}\rangle = \texttt{CU}$.

Formalizing Contexts

A context is a term with a hole? Not really...

For $C := \lambda x.\langle\cdot\rangle$, we have that $C\langle x\rangle = \lambda x.x$ and $C\langle y\rangle = \lambda x.y$

A context is a term with a hole *annotated by the possible captured variables*...

$$\lambda x.\Box_x$$

A context is a function from `tm` to `tm`, coded in Abella as a predicate:

`ctx T CT` holds iff there exists a context C such that $C\langle \texttt{T}\rangle = \texttt{CT}$.

Applying a context to two different terms:

`ctxs T CT U CU` holds iff there exists a context C such that $C\langle \texttt{T}\rangle = \texttt{CT}$ and $C\langle \texttt{U}\rangle = \texttt{CU}$.

Takahashi's Trick in CbN

$$C\langle u\rangle \leftrightarrow t_C\{x{\leftarrow}u_C\}$$
$$C\langle s\rangle \leftrightarrow t_C\{x{\leftarrow}s_C\}$$

Trick:
Let $\mathtt{fv}(u) \cup \mathtt{fv}(s) = \{x_1, \ldots, x_k\}$, and y a fresh variable.

- $u_C := \lambda x_1 \ldots \lambda x_k.u$ and $s_C := \lambda x_1 \ldots \lambda x_k.s$ are closed terms.
- Consider $t_C := C\langle y x_1 \ldots x_k\rangle$, and note that:

$$\begin{array}{rcc} t_C\{y{\leftarrow}u_C\} & = & C\langle u_C x_1 \ldots x_k\rangle \\ & = & C\langle (\lambda x_1 \ldots \lambda x_k.u) x_1 \ldots x_k\rangle \\ & \to_\beta^k & C\langle u\rangle \end{array}$$

- u is h-diverging implies that u_C is also h-diverging.
- $C\langle u\rangle$ is h-normalizing if and only if $t\{y{\leftarrow}u_C\}$ is. (also true for s and s_C)

  by the Head Normalization Theorem (and confluence, etc.)

Formalizing Takahashi's Trick

Lemma (Disentangling)

Let C be a context. Then there exist t_C and a variable $x \notin \mathtt{fv}(C)$ such that for all terms u there exists u_C such that $t_C\{x \leftarrow u_C\} \to_\beta^* C\langle u \rangle$. Moreover, if u is head divergent then u_C is head divergent.

Some technicalities in Abella...

Definition (Substitution preorder)

The *substitution preorder* $t \precsim_{\mathcal{S}}^{\mathtt{h}} u$ holds when $s\{x \leftarrow \lambda y_1 \ldots \lambda y_n.t\}$ $\to_{\mathtt{h}}$-terminating implies that $s\{x \leftarrow \lambda y_1 \ldots \lambda y_n.u\}$ is $\to_{\mathtt{h}}$-terminating for all terms s, variables x, and lists of variables $y_1, \ldots, y_n$ with $n \geq 0$.

Back to Light Genericity

We can show (in Abella) light and heavy genericity using this disentangling lemma.

We also show that the head contextual preorder includes β-conversion.

Relies on head normalization, confluence, etc.

Which lead us to:

- The consistency of collapsing $\mathtt{h}$-diverging terms, and;
- The maximality of the head contextual preorder.

Back to Light Genericity

We can show (in Abella) light and heavy genericity using this disentangling lemma.

We also show that the head contextual preorder includes β-conversion.

Relies on head normalization, confluence, etc.

Which lead us to:

- The consistency of collapsing $\mathtt{h}$-diverging terms, and;
- The maximality of the head contextual preorder.

Back to Light Genericity

We can show (in Abella) light and heavy genericity using this disentangling lemma.

We also show that the head contextual preorder includes β-conversion.

Relies on head normalization, confluence, etc.

Which lead us to:

- The consistency of collapsing `h`-diverging terms, and;
- The maximality of the head contextual preorder.

Constructive Contextual Equivalence?

The proof of maximality always starts by:

If $\mathcal{T} \vdash t \leq u$ and $t \not\preceq^{\mathrm{octx}}_{\mathtt{h}} u$
Then $\underline{\exists C}$ such that

- $C\langle t \rangle$ is $\mathtt{h}$-normalizing and
- $C\langle u \rangle$ is $\mathtt{h}$-diverging.

In general, not valid in intuistionistic logic

$$\neg\forall\phi \not\Rightarrow \exists\neg\phi$$

Constructive Contextual Equivalence?

The proof of maximality always starts by:

If $\mathcal{T} \vdash t \leq u$ and $t \not\preceq^{\mathrm{octx}}_{\mathtt{h}} u$
Then $\underline{\exists C}$ such that

- $C\langle t \rangle$ is $\mathtt{h}$-normalizing and
- $C\langle u \rangle$ is $\mathtt{h}$-diverging.

In general, not valid in intuistionistic logic

$$\neg\forall\phi \not\Rightarrow \exists\neg\phi$$

Why I think everyone[1] should start using Abella

and why I might stop using it

- Easy to get started with, only few tactics
- Pen-and-paper proofs translate easily

- Expressive power is limited (I would like to quantify over relations)
- Missing some "quality of life" features

  Reasoning with integers can already be painful...

[1] in the comprehensive set of people interested in functional languages with binders

Conclusions

- Towards a Formal Companion of Barendregt's book
- Easy proofs that rely mostly on rewriting/operational results
- Formalization faithful to pen-and-paper proofs

Future work:

- Can be adapted to CbV but results are not formalized (also a lot of the results can be factorized)
- Constructive Contextual (In)Equivalence?
- Other results on program equivalence to be made formal (mechanizing Böhm trees and Böhm's theorem?)

Thank you!

Conclusions

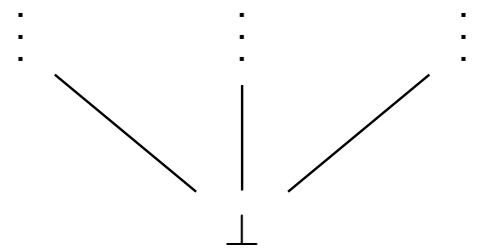
- Towards a Formal Companion of Barendregt's book
- Easy proofs that rely mostly on rewriting/operational results
- Formalization faithful to pen-and-paper proofs

Future work:

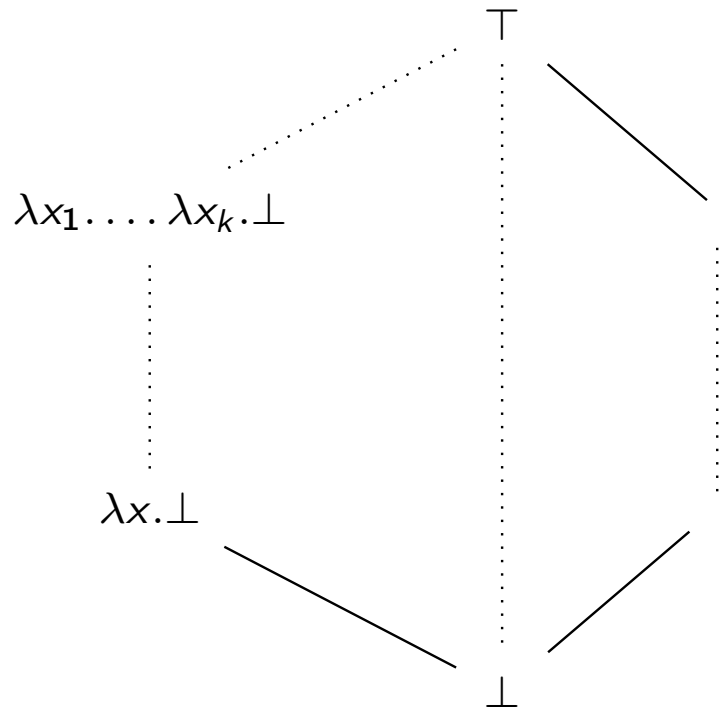
- Can be adapted to CbV but results are not formalized (also a lot of the results can be factorized)
- Constructive Contextual (In)Equivalence?
- Other results on program equivalence to be made formal (mechanizing Böhm trees and Böhm's theorem?)

Thank you!

Thank you!



CbN



CbV

Contextual preorder for lambda terms

$\bot :=$ equivalence class of Ω

Formalizing λ

$$\text{Terms} \quad t, u \quad := \quad x \mid \lambda x.t \mid tu$$

λ-terms and the predicate for inducting on them in Abella:

```
Kind  tm   type.

Type  abs  (tm -> tm) -> tm.
Type  app  tm -> tm -> tm.


Define is_tm : tm -> prop by
  nabla x, is_tm x;
  is_tm (abs T) := nabla x, is_tm (T x);
  is_tm (app T U) := is_tm T /\ is_tm U.
```

Formalizing λ

$$\text{Terms} \quad t, u \quad := \quad x \mid \lambda x.t \mid tu$$

λ-terms and the predicate for inducting on them in Abella:

```
Kind  tm   type.

Type  abs  (tm -> tm) -> tm.
Type  app  tm -> tm -> tm.


Define is_tm : tm -> prop by
  nabla x, is_tm x;
  is_tm (abs T) := nabla x, is_tm (T x);
  is_tm (app T U) := is_tm T /\ is_tm U.
```

Formalizing λ and β

$$\frac{}{(\lambda x.t)u \mapsto_\beta t\{x \leftarrow u\}} \qquad \frac{t \to_\beta t'}{tu \to_\beta t'u} \qquad \frac{t \to_\beta t'}{\lambda x.t \to_\beta \lambda x.t'} \qquad \frac{u \to_\beta u'}{tu \to_\beta tu'}$$

```
Define beta : tm -> tm -> prop by
  beta (app (abs T) U) (T U);
  beta (app T U) (app T' U) := beta T T';
  beta (abs T) (abs T') := nabla x, beta (T x) (T' x);
  beta (app T U) (app T U') := beta U U'.
```

$$\frac{}{(\lambda x.t)u \to_{\mathrm{h}} t\{x \leftarrow u\}} \qquad \frac{t \to_{\mathrm{h}} u}{ts \to_{\mathrm{h}} us} \qquad \frac{t \to_{\mathrm{h}} u}{\lambda x.t \to_{\mathrm{h}} \lambda x.u}$$

Formalizing λ and β

$$\frac{}{(\lambda x.t)u \mapsto_\beta t\{x \leftarrow u\}} \qquad \frac{t \to_\beta t'}{tu \to_\beta t'u} \qquad \frac{t \to_\beta t'}{\lambda x.t \to_\beta \lambda x.t'} \qquad \frac{u \to_\beta u'}{tu \to_\beta tu'}$$

```
Define beta : tm -> tm -> prop by
  beta (app (abs T) U) (T U);
  beta (app T U) (app T' U) := beta T T';
  beta (abs T) (abs T') := nabla x, beta (T x) (T' x);
  beta (app T U) (app T U') := beta U U'.
```

$$\frac{}{(\lambda x.t)u \to_{\mathrm{h}} t\{x \leftarrow u\}} \qquad \frac{t \to_{\mathrm{h}} u}{ts \to_{\mathrm{h}} us} \qquad \frac{t \to_{\mathrm{h}} u}{\lambda x.t \to_{\mathrm{h}} \lambda x.u}$$

Formalizing λ and β

$$\frac{}{(\lambda x.t)u \mapsto_\beta t\{x \leftarrow u\}} \qquad \frac{t \to_\beta t'}{tu \to_\beta t'u} \qquad \frac{t \to_\beta t'}{\lambda x.t \to_\beta \lambda x.t'} \qquad \frac{u \to_\beta u'}{tu \to_\beta tu'}$$

```
Define beta : tm -> tm -> prop by
  beta (app (abs T) U) (T U);
  beta (app T U) (app T' U) := beta T T';
  beta (abs T) (abs T') := nabla x, beta (T x) (T' x);
  beta (app T U) (app T U') := beta U U'.
```

$$\frac{}{(\lambda x.t)u \to_{\mathtt{h}} t\{x \leftarrow u\}} \qquad \frac{t \to_{\mathtt{h}} u}{ts \to_{\mathtt{h}} us} \qquad \frac{t \to_{\mathtt{h}} u}{\lambda x.t \to_{\mathtt{h}} \lambda x.u}$$

Formalizing λ-theories

A λ-theory is stable by contexts...

Contextual equivalence...

We need to formalize contexts

Formalizing λ-theories

A λ-theory is stable by contexts...

Contextual equivalence...

We need to formalize contexts

Formalizing λ-theories

A λ-theory is stable by contexts...

Contextual equivalence...

We need to formalize contexts

Formalizing Contexts

A context is a term with a hole? Not really...

Set $C := \lambda x.\langle\cdot\rangle$, then $C\langle y\rangle = \lambda x.y$ and $C\langle x\rangle = \lambda x.x = \mathtt{I}$.

`ctx T CT` holds iff there exists a context C such that $C\langle \mathtt{T}\rangle = \mathtt{CT}$.

```
Define ctx : tm -> tm -> prop by
  ctx T T;
  ctx T (app P Q) := ctx T P \/ ctx T Q;
  nabla x, ctx (T x) (abs CT) :=
    nabla x, ctx (T x) (CT x).
```

How to apply a context to two different terms?

Formalizing Contexts

A context is a term with a hole? Not really...
Set $C := \lambda x.\langle\cdot\rangle$, then $C\langle y\rangle = \lambda x.y$ and $C\langle x\rangle = \lambda x.x = \mathtt{I}$.

`ctx T CT` holds iff there exists a context C such that $C\langle \mathtt{T}\rangle = \mathtt{CT}$.

```
Define ctx : tm -> tm -> prop by
  ctx T T;
  ctx T (app P Q) := ctx T P \/ ctx T Q;
  nabla x, ctx (T x) (abs CT) :=
    nabla x, ctx (T x) (CT x).
```

How to apply a context to two different terms?

Formalizing Contexts

A context is a term with a hole? Not really...
Set $C := \lambda x.\langle\cdot\rangle$, then $C\langle y\rangle = \lambda x.y$ and $C\langle x\rangle = \lambda x.x = \mathtt{I}$.

`ctx T CT` holds iff there exists a context C such that $C\langle \mathtt{T}\rangle = \mathtt{CT}$.

```
Define ctx : tm -> tm -> prop by
  ctx T T;
  ctx T (app P Q) := ctx T P \/ ctx T Q;
  nabla x, ctx (T x) (abs CT) :=
    nabla x, ctx (T x) (CT x).
```

How to apply a context to two different terms?

Formalizing Contexts

A context is a term with a hole? Not really...
Set $C := \lambda x.\langle\cdot\rangle$, then $C\langle y\rangle = \lambda x.y$ and $C\langle x\rangle = \lambda x.x = \mathtt{I}$.

`ctx T CT` holds iff there exists a context C such that $C\langle \mathtt{T}\rangle = \mathtt{CT}$.

```
Define ctx : tm -> tm -> prop by
  ctx T T;
  ctx T (app P Q) := ctx T P \/ ctx T Q;
  nabla x, ctx (T x) (abs CT) :=
    nabla x, ctx (T x) (CT x).
```

How to apply a context to two different terms?

Formalizing Contexts

A context is a term with a hole? Not really...
Set $C := \lambda x.\langle\cdot\rangle$, then $C\langle y\rangle = \lambda x.y$ and $C\langle x\rangle = \lambda x.x = \mathtt{I}$.

`ctx T CT` holds iff there exists a context C such that $C\langle \mathtt{T}\rangle = \mathtt{CT}$.

```
Define ctx : tm -> tm -> prop by
  ctx T T;
  ctx T (app P Q) := ctx T P \/ ctx T Q;
  nabla x, ctx (T x) (abs CT) :=
    nabla x, ctx (T x) (CT x).
```

How to apply a context to two different terms?

Formalizing Contexts

`ctxs T CT U CU` holds
iff there exists a context C such that $C\langle \texttt{T} \rangle = \texttt{CT}$ and $C\langle \texttt{U} \rangle = \texttt{CU}$.

```
Define ctxs : tm -> tm -> tm -> tm -> prop by
  ctxs T T U U;
  ctxs T (app A B) U (app C D) :=
    (ctxs T A U C /\ B = D /\ tm D)
    \/ (ctxs T B U D /\ A = C /\ tm C);
  nabla x, ctxs (T x) (abs CT) (U x) (abs CU):=
    nabla y, ctxs (T y) (CT y) (U y) (CU y).
```

Formalizing Contexts

`ctxs T CT U CU` holds
iff there exists a context C such that $C\langle \texttt{T} \rangle = \texttt{CT}$ and $C\langle \texttt{U} \rangle = \texttt{CU}$.

```
Define ctxs : tm -> tm -> tm -> tm -> prop by
  ctxs T T U U;
  ctxs T (app A B) U (app C D) :=
    (ctxs T A U C /\ B = D /\ tm D)
    \/ (ctxs T B U D /\ A = C /\ tm C);
  nabla x, ctxs (T x) (abs CT) (U x) (abs CU):=
    nabla y, ctxs (T y) (CT y) (U y) (CU y).
```

Formalizing Contextual Preorder

```
Define ctx_preord : tm -> tm -> prop by
  ctx_preord P Q := forall CP CQ,
    tm P -> tm Q ->
    ctxs P CP Q CQ -> head_terminating CP ->
    head_terminating CQ.
```

- `ctx_preord` is stable by contexts.
- `ctx_preord` is invariant under computation.
- `ctx_preord` has `h`-diverging terms as minimums.

Light Genericity

Light Genericity: head-diverging terms are minimum for the head open contextual preorder.

Unfolded statement:

Light Genericity: let u be head-diverging and C such that $C\langle u\rangle$ is head-normalizing then $C\langle t\rangle$ is head-normalizing for all $t \in \Lambda$.

Main difficulty: reasoning with contexts and reduction.

Light Genericity

Light Genericity: head-diverging terms are minimum for the head open contextual preorder.

Unfolded statement:

Light Genericity: let u be head-diverging and C such that $C\langle u\rangle$ is head-normalizing then $C\langle t\rangle$ is head-normalizing for all $t \in \Lambda$.

Main difficulty: reasoning with contexts and reduction.

Light Genericity

Light Genericity: head-diverging terms are minimum for the head open contextual preorder.

Unfolded statement:

Light Genericity: let u be head-diverging and C such that $C\langle u\rangle$ is head-normalizing then $C\langle t\rangle$ is head-normalizing for all $t \in \Lambda$.

Main difficulty: reasoning with contexts and reduction.

Direct proof of Light Genericity

Takahashi proves Barendregt's heavy genericity with a very short proof [Tak94] and gives as a corollary light genericity.

Key idea/trick: Reason with substitutions instead of contexts!

Light genericity as substitution: let u be $\mathtt{h}$-diverging and t such that $t\{x \leftarrow u\}$ is $\mathtt{h}$-normalizing then $t\{x \leftarrow s\}$ is $\mathtt{h}$-normalizing for all $s \in \Lambda$.

Direct proof of Light Genericity

Takahashi proves Barendregt's heavy genericity with a very short proof [Tak94] and gives as a corollary light genericity.

Key idea/trick: Reason with substitutions instead of contexts!

Light genericity as substitution: let u be $\mathtt{h}$-diverging and t such that $t\{x \leftarrow u\}$ is $\mathtt{h}$-normalizing then $t\{x \leftarrow s\}$ is $\mathtt{h}$-normalizing for all $s \in \Lambda$.

Direct proof of Light Genericity

Takahashi proves Barendregt's heavy genericity with a very short proof [Tak94] and gives as a corollary light genericity.

Key idea/trick: Reason with substitutions instead of contexts!

Light genericity as substitution: let u be $\mathtt{h}$-diverging and t such that $t\{x \leftarrow u\}$ is $\mathtt{h}$-normalizing then $t\{x \leftarrow s\}$ is $\mathtt{h}$-normalizing for all $s \in \Lambda$.

Takahashi's Trick in CbN

$$C\langle u\rangle \leftrightarrow t_C\{x{\leftarrow}u_C\}$$
$$C\langle s\rangle \leftrightarrow t_C\{x{\leftarrow}s_C\}$$

Trick:
Let $\mathtt{fv}(u) \cup \mathtt{fv}(s) = \{x_1, \ldots, x_k\}$, and y a fresh variable.

- $u_C := \lambda x_1 \ldots \lambda x_k.u$ and $s_C := \lambda x_1 \ldots \lambda x_k.s$ are closed terms.
- Consider $t_C := C\langle y x_1 \ldots x_k\rangle$, and note that:

$$\begin{array}{rcc} t_C\{y{\leftarrow}u_C\} & = & C\langle u_C x_1 \ldots x_k\rangle \\ & = & C\langle (\lambda x_1 \ldots \lambda x_k.u) x_1 \ldots x_k\rangle \\ & \to_\beta^k & C\langle u\rangle \end{array}$$

- u is h-diverging implies that u_C is also h-diverging.
- $C\langle u\rangle$ is h-normalizing if and only if $t\{y{\leftarrow}u_C\}$ is. (also true for s and s_C)

  by the Head Normalization Theorem (and confluence, etc.)

Conclusions

- A small subset of Barendregt's book formalized (many rewriting theorems hidden in this presentation)
- Easy proofs that rely mostly on rewriting/operational results
- Faithful formalization of the pen-and-paper proofs

Future work:

- Constructive Contextual (In)Equivalence?
- Many results adapt to the theory of the Call-by-Value calculus (haven't formalized these)
- Other results on program equivalence to be made formal (mechanizing Böhm trees and Böhm's theorem?
  $\Rightarrow$ intensional presentation of contextual equivalence)

Thank you!

Conclusions

- A small subset of Barendregt's book formalized (many rewriting theorems hidden in this presentation)
- Easy proofs that rely mostly on rewriting/operational results
- Faithful formalization of the pen-and-paper proofs

Future work:

- Constructive Contextual (In)Equivalence?
- Many results adapt to the theory of the Call-by-Value calculus (haven't formalized these)
- Other results on program equivalence to be made formal (mechanizing Böhm trees and Böhm's theorem?
  $\Rightarrow$ intensional presentation of contextual equivalence)

Thank you!


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.


Thomas Ehrhard.
Collapsing non-idempotent intersection types.
In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 259–273. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.


James Hiram Morris.
Lambda-calculus Models of Programming Languages.

PhD thesis, Massachusetts Institute of Technology, 1968.


Gordon D. Plotkin.
Call-by-name, call-by-value and the λ-calculus.
Theoretical Computer Science, 1(2):125–159, 1975.


Masako Takahashi.
A simple proof of the genericity lemma, pages 117–118.
Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.