10-page summary of ADRIENNE LANCELOT's PhD thesis titled

COMPARING FUNCTIONAL PROGRAMS, OR HOW TO PUT λ-TERMS BACK IN ORDER

A THEORY OF PROGRAM EQUIVALENCES AND PREORDERS THROUGH MEANING, EVALUATION ORDERS AND COSTS

Thesis defended on November 13th 2025

Available on HAL [Lancelot, 2025a]

This thesis investigates the theory of programming languages, and in particular the concept of program equivalence in the λ-calculus, the mathematical core of functional programming. This summary is organized as a gentle introduction to the main concepts, for the first four pages, followed by an overview of the three parts of the thesis and their main contributions.

This thesis is based on five PUBLICATIONS: in FoSSaCS 2024 [Accattoli and Lancelot, 2024a], FSCD 2024 [Accattoli and Lancelot, 2024b], POPL 2025 [Accattoli, Lancelot, Manzonetto, et al., 2025], post-proceedings of TYPES 2025 [Lancelot, 2025b] and in ITP 2025 [Lancelot et al., 2025], and extended versions available on ArXiv or HAL as well as an unpublished technical report available on ArXiV [Accattoli, Lancelot, and Faggian, 2023].

SUMMARY

Programming languages define the grammar of accepted program phrases used by programmers to interact with computing hardware. The meaning of these syntactic instructions available to the programmer are provided by *semantics*. Most semantics almost read as tautologies: the $+$ sign of the language represents the addition of two integers. In fact, as computers have finite memory, this specification of the implementation of $+$ may be misleading; in most languages, there indeed exists two integers n and m such that $n + m$ does not compute to the mathematical addition of the two integers because it would be a number too large for the computer representation.

Beyond the informal description of semantics in documentations of programming languages, semanticists study and provide mathematical specifications of semantics. They exist in two main flavors: *operational* and *denotational*.

OPERATIONAL A compiler provides, in a way, the first semantics of a language, by compiling the language syntax into another language (of which we know the semantics). Operational semantics define how to execute programs (by compilation, rewriting, abstract machines, etc.).

DENOTATIONAL The study of semantic models is about building interpretations of syntactic programs into mathematical spaces. They aim to use these mathematics to answer safety questions (correctness of compiler transformations, ensuring error-free termination) and to provide proofs that a program phrase exactly matches its intended use case, formally written as its *specification* and represented as a point in the mathematical model.

All programs with the same specification are equivalent to the semanticist eye, as they aim to prove that the model represents specifications and not exact program phrases.

Computable Functions. One has to be careful when picking a mathematical space to study programs as not everything is computable, as was shown by Turing's halting problem. A key issue in mathematical semantics of programs is that, while syntax can only describe computable functions–which is basically the content of Church's Thesis–interpretations in mathematical models may be contained in a richer world, where non-computable objects also exists. The function space $\mathbb{N} \to \mathbb{N}$ includes non-computable functions such as the halting problem (as termination is undecidable). Mathematical function spaces such as Scott domains may fail to exclude the parallel-or connective, which is famously not computable by a sequential program.

To develop programming language semantics, one needs to pick a computable basis to model. One of the first such computable set to be described in the literature is that of *partial recursive functions* (shortened to PRFs). PRFs represent *computable* functions from $\mathbb{N}$ to $\mathbb{N}$, that is functions that can be computed by a program of type $\mathbb{N} \to \mathbb{N}$. They are mostly studied in a mathematical flavor (rather than in an *operational* or computational one) and hence are by definition only an extensional representation of a program: there is no way to access the source code, only a description of the resulting output (or lack thereof) when given a specific input.

Based on the *partiality* of these PRFs, there exists a natural way to compare PRFs: the undefined value $\bot$ induces an extensional (pre)order $f \leq_{\text{PRF}} g$ holding when $\forall n \in \mathbb{N}$, either

The dynamics of computation are absent in PRFs, making it easy to produce an extensional preorder but not quite possible to compare programs intensionally.

$f(n) = \bot$ or $f(n) =_{\mathbb{N}} g(n)$. The everywhere undefined function $f_\bot(n) \coloneqq \bot$ is the minimum element for this preorder.

The λ-Calculus. Church's λ-calculus is the paradigmatic mathematical programming language. The λ-calculus, contrarily to PRFs, is all about the *computation* procedure: programs are represented with syntax trees and their evaluation is given step by step via a *rewriting* system.

PRFs and the λ-calculus can represent each other [Barendregt, 1984, Chapter 6]. Moving to the λ-calculus provides dynamic computations, which yield different syntactic implementations of functions that—extensionally—display the same behavior. Barendregt's theory of the λ-calculus is based on the idea of studying the semantics—that is, the equational theory—induced on λ-terms by the representation of the extensional order on PRFs.

Notions of Program Equivalence. This thesis revolves around the notion of program equivalence: what does it mean for two programs (in this thesis: two λ-terms) to be equivalent?

We first give a broad idea of the main identifications of programs one may want to include in a notion of program equivalence (with dummy code examples):

1. Are they written with similar phrasing? By similar, we can think of the exact same, which is reasonable as `print("Hello World")` is clearly equivalent to itself, but it should also be equivalent to `print("Hello"+" World")`. While these two programs do not have exactly the same syntax, if one evaluates `"Hello"+" World"` one should get `"Hello World"` as a result and so the `print` function is actually called on the same argument in both cases. We therefore in general wish our program equivalences to be *invariant under computation*, computation rules being specified for a language as its *operational semantics* backbone.

   Invariance under computation.

2. Does their meaning unfold to the same object? Not all interesting programs are computable to a finite result, sometimes the evaluation diverges–but still produces increasing information. We consider a dummy example where these two programs cannot be easily identified only with computational invariance. Consider the two programs `while(true){ print("a");}` and `while(true){ print("a"); print("a")}`: to identify them (in a non-adhoc way), one must build an infinite structure `print("a"); print("a"); print("a");...` and show that both programs generate such an execution trace.

   The key example is the identification of different implementations of recursion (e. g. non β-convertible fixed-point combinators), usually shown by equality of their Böhm trees.

   Böhm tree-like identifications.

3. We might want to take into account the cost of a program: a more efficient program can sometimes do the job just fine. Cost-efficiency may mean different things depending on the notion of cost. Time of execution is a reasonable benchmark to try to speed up programs, as well as the space needed to compute to reduce the memory used up by programs, and perhaps more generally accesses to external resources (global memory cells, connections to internet servers, etc.). Our aim is not to compute the exact cost of programs, but rather to take into account these costs when comparing two programs.

   Such inspection may lead to looking at program phrases in detail, as denotational semantics tends to abstract away from code and implementation details, potentially hiding execution costs.

   Quantitative program equivalences.

CONTEXTUAL EQUIVALENCE

The starting point and one of the focus of this thesis is the standard notion of *contextual equivalence*, introduced by Morris [1968] to study the untyped λ-calculus. Contextual equivalence,

for a chosen predicate $\Downarrow_{\mathtt{obs}}$ of *observation* (typically some notion of termination), is defined as testing over all possible execution environments:

$$t \equiv^{\mathtt{ctx}}_{\mathtt{obs}} u \quad \text{if for all contexts } C\,. \quad [\, C\langle t\rangle \Downarrow_{\mathtt{obs}} \quad \Leftrightarrow \quad C\langle u\rangle \Downarrow_{\mathtt{obs}} \,]$$

One of the main protagonists of this thesis, in fact one of the most studied contextual equivalence, is *head contextual equivalence* $\equiv^{\mathtt{ctx}}_{\mathtt{h}}$ for the untyped λ-calculus [Barendregt, 1984], where one observes termination of head reduction $\to_{\mathtt{h}}$.

Contextual Indistinguishability. Contextual equivalence embodies a *black box* behavioral principle with respect to termination. Nothing is said about the structure of t and u, it is only prescribed that any use of t in a larger program can be replaced by u (and vice-versa) without affecting the observables.

The general and natural aspects of the definition of contextual equivalence make it one of the most appealing notion of program equivalence, both for semanticists and in more applied settings. The typical example being program transformations at work in compilers, for which contextual equivalence guarantees a strong form of soundness [Patrignani et al., 2019].

A contextual *preorder* $t \sqsubseteq^{\mathtt{ctx}}_{\mathtt{obs}} u$ is easily defined as an asymmetric variant (by switching the "if and only if" $\Leftrightarrow$ to an "implies" $\Rightarrow$). Contextual preorders are more lax than their equivalence counterpart: they only require that any use of t in a larger program can be replaced by u ~~(and vice-versa)~~ without affecting the observables. In practice, this means that u could be able to do more than t but it should not break the specification laid out by the original program t.

Is It Tractable? A potential downside of Morris' contextual equivalence [1968] is that it is difficult to prove the equivalence of two terms, because of the universal quantification over contexts. Other candidates, such as Abramsky's applicative bisimilarity [1990] can also be challenging to establish, especially for recursive programs.

To provide tractable proof techniques for contextual equivalence, we define *normal form bisimilarities* and prove that they entail contextual equivalence (several of which are defined in this thesis and especially in PART III).

Equational Contextual Equivalence. In order for contextual equivalence $\equiv^{\mathtt{ctx}}_{\mathtt{obs}}$ to be a proper semantical notion, one actually has to show that it is an *equational theory* for $\mathcal{L}$. This means that $\equiv^{\mathtt{ctx}}_{\mathtt{obs}}$ must be *compatible*, i. e. stable under context closure, and *invariant under computation*, i. e. $\equiv^{\mathtt{ctx}}_{\mathtt{obs}}$ must include the conversion of the language (β-conversion $=_\beta$ for the λ-calculus).

While compatibility holds by definition, invariance is a non-trivial theorem, because of the universal quantification on contexts in $\equiv^{\mathtt{ctx}}_{\mathtt{obs}}$, which is hard to manage in proofs. For head contextual equivalence $\equiv^{\mathtt{ctx}}_{\mathtt{h}}$, for instance, a rewriting-based proof of the invariance property requires to prove the confluence of β-reduction and various properties of head reduction.

An equational theory for $\mathcal{L}$ can be seen as a *semantics* for $\mathcal{L}$. The abstract requirement defining denotational models is that their induced equivalence[1] is an equational theory. Notably, head contextual equivalence $\equiv^{\mathtt{ctx}}_{\mathtt{h}}$ is exactly the equational theory of Scott's $\mathcal{D}_\infty$ model [1972], historically the first model of the untyped λ-calculus.

The theoretical relevance of contextual equivalence shines through when it is presented as an equational theory. Indeed, (for the right notion of observables) it often is a maximal consistent equational theory, that is, it is pointless to study an equational theory including contextual equivalence and a new equation $(t = u)$, as this equational theory equates all terms. We study in depth the maximality principle in this thesis.

1 The equivalence induced by a model $\mathcal{M}$ is defined as $t =_{\mathcal{M}} u$ if $[\![t]\!] = [\![u]\!]$, where $[\![\cdot]\!]$ is the interpretation in the model.

What About Optimizations? Contextual equivalence is an extensional principle, akin to testing every program with every possible input, but it does not check whether equivalent programs have a comparable cost (time of execution, memory space required to compute, etc.). PART II introduces a notion of quantitative contextual equivalence based on counting interactions between a program and its execution environment.

EVALUATION PARADIGMS, OR HOW TO COMPUTE?

Evaluation in the λ-calculus comes in various flavors, just like there are different ways to compute mathematical expressions. The study of programming languages, and especially rewriting theory, focuses on giving a mathematical first-class status to *how to* compute, whereas mathematicians in general do not care about the process but only the result of a computation.

Call-by-Name vs. Call-by-Value. The two main evaluation paradigms are call-by-name and call-by-value. The former is Church's original presentation and has been the subject of an extensive theoretical treatment. The gap from theory and practice enters the room: call-by-name is inefficient by design (duplicating computations) so most programming languages do not implement call-by-name evaluation. Call-by-value evaluation aims to not repeat the same computations, making the execution time more appropriate for programming. Still, the theory of call-by-value is more subtle, as is shown in this thesis, and had evaded a theoretical treatment until the last decade or so.

Comparing Call-by-Name and Call-by-Value Program Equivalences. The gap between call-by-name and call-by-value evaluation is not only about efficiency: it also changes which programs terminate, i. e. for which the rewriting procedure finds a result. The fact that call-by-name is not equivalent in terms of termination to call-by-value means that CbN and CbV contextual equivalences are different.

There are other evaluation paradigms, the most well-known is *Call-by-Need* (CbNeed), which combines the efficiency of call-by-value and the erasure principle of call-by-name. It is discussed in the thesis to introduce and motivate its informal dual, the *Call-by-Silly* (CbS) variant, which combines the inefficiency of call-by-name and the non-erasure principle of call-by-value. Regarding (effect-free) contextual equivalence, it is well known that CbN and CbNeed induce the same contextual equivalence [Ariola et al., 1995; Kesner, 2016] and it turns out that CbV and CbS have the same contextual equivalence (PART III & Accattoli and Lancelot [2024b]).

THREE BROAD CONTRIBUTIONS OF THIS THESIS

This thesis explores program equivalences for functional languages, focusing on the λ-calculus. There are three main broad ideas that we highlight briefly:

ORDERS ON λ-TERMS: we focus on program preorders and *in*equational theories. We study what these orders on λ-terms yield and the notion of minimum and maximum terms.

INTERACTION EQUIVALENCE: we introduce a new quantitative refinement of contextual equivalence, by introducing a syntactic internal/external distinction.

CALL-BY-VALUE PROGRAM EQUIV.: we provide an overview of possible equational theories for call-by-value λ-calculi, where the landscape is much richer than for call-by-name.

We overview the precise contributions in the next three sections of this document.

The Theory of the λ-Calculus. Before going more in depth, let us discuss broadly the theory of the λ-calculus. Barendregt's book [1984] brings together most results developed until the eighties. The λ-calculus as a model of computation is the starting point of Barendregt's book with the central concept of solvability, motivated by the representation of partial recursive functions into the λ-calculus. Barendregt and Manzonetto's sequel book [2022] still gathers results on the theory of the untyped λ-calculus, answering some open questions from the first book. Therein, the focus is perhaps less motivated by the questions of solvability and PRFs but rather to complete the theory of the λ-calculus as a first-class mathematical object on its own.

Meaningless Terms in Call-by-Name. Meaningless terms were first studied in the 1970s, by Wadsworth [1971, 1976] and Barendregt [1971, 1977a], while working on denotational models and the representation of partial recursive functions. Meaningless terms are meant to represent the *being undefined* of PRFs—considered as the paradigmatic meaningless computation—and end up being characterized as *unsolvable* terms. Barendregt's book [1984] is built around the concept of (un)solvability. An important point is that meaningless terms can be identified in equational theories (and are indeed identified by contextual equivalence), as it is not possible to distinguish meaninglessness.

The Theory of the Call-by-Value λ-Calculus. This thesis still aims to contribute to the study of the theory of the λ-calculus, and especially in the theory of call-by-value. Key results about theories and contextual equivalence already appear in [Accattoli and Guerrieri, 2022] but not all CbN results are replicated in the CbV λ-calculus. A missing piece pointed out by Accattoli and Guerrieri is a CbV *genericity lemma*, which is critical in representing undefinedness of PRFs and the extensional PRF preorder $\leq_{\text{PRF}}$ inside the λ-calculus.

Meaningless Terms in Call-by-Value. Some of the work presented in this thesis stems from the desire to reproduce the analysis of meaningless terms for the untyped *call-by-value* λ-calculus (PART III). Perhaps surprisingly, the call-by-value (shortened to CbV) λ-calculus behaves quite differently with respect to meaningless terms. Accattoli and Guerrieri's recent study of meaningless terms in CbV [2022] indeed stresses two key differences: *genericity fails in CbV*, and *collapsibility fails as well*, as any equational theory equating CbV meaningless terms is *inconsistent*, if one considers as meaningless the CbV analogous of unsolvable terms. Accattoli and Guerrieri also show that collapsibility can be recovered by adopting a different notion of meaningless terms, namely *CbV inscrutable terms*, but they do not prove genericity for them.

In this thesis, we do prove a genericity result for inscrutable terms, and also provide a new proof of their collapsibility. These results, however, are only a small part of the contributions of this study of (in)equational theories and program equivalences.

ORDERS ON λ-TERMS (*this section describes the main contributions of* PART I)

The first chapters of this thesis contain *revisited* preliminaries: we introduce the necessary background with refreshed proofs and abstracting over the evaluation paradigm considered.

We develop a uniform theory for program preorders and equivalences. The refinement from equational theories to their asymmetric counterpart, *inequational theories*, allows us to rephrase cornerstone results of Barendregt's meta-theory as basic properties over program preorders.

We study preorders *because relations over λ-terms are rarely antisymmetric or total orders.*

Minimum & Maximum Terms. Studying orderings in particular makes it possible to study minimum and maximum programs. Minimum pieces of code are those that can be replaced by anything else without changing the overall use cases of the code. On the other side, maximum programs are those that can simulate any piece of code, they can be seen as a strong notion

of terminating computation, or a *happy client* (the program reaches a special terminating and accepting state which only propagates to larger programs, it cannot turn back to a typical program that could result in an error).

IN THIS THESIS we provide characterizations of extremum terms for different λ-calculi and show the theoretical relevance of these characterizations as extremums of meaningless/super meaningful terms. The study of minimum terms for the contextual preorder ended up simplifying a classic result in the theory of the λ-calculus, namely Barendregt's genericity result [Barendregt, 1984, Prop. 14.3.24], to a new *light genericity* statement. The study of *maximum* terms created a new *co-genericity* statement that led to a new result in the theory of solvability in the call-by-value λ-calculus.

Maximality of contextual equivalence. Contextual equivalence tends to equate many terms, usually identifying much more than computational equivalence (only based on the computations of the underlying programming language). In fact, it is often not consistent to equate more terms.

Theorem (**1.4.3** at p.57 ∈ Lancelot 2025a)**.** *Under some assumptions on the evaluation strategy* $\mathtt{s}$ *and when observing termination for the strategy* $\mathtt{s}$*, contextual equivalence* $\equiv^{\mathsf{ctx}}_{\mathsf{obs}}$ *is* maximal*: any equational theory larger than* $\equiv^{\mathsf{ctx}}_{\mathsf{obs}}$ *equates all programs together.*

The proof provided in this thesis is not the one from Barendregt's book [1984]: it is simple, using only light genericity, and unifying, providing sufficient conditions to prove maximality across different evaluation paradigms (namely call-by-name and call-by-value).

Formalizing the Revisited Theory. The results for call-by-name evaluation are formalized in the Abella proof assistant [Lancelot et al., 2025], faithfully mimicking pen-and-paper proofs. Formalizing these results constitutes an important first step toward a *fully formalized theory of the λ-calculus*, refreshing and complementing Barendregt's reference book [1984].

The mechanization is only commented on in the thesis and not explained in detail.

This work revisits foundational results in the theory of the λ-calculus, but we do not use the foundational proofs, in particular none of Barendregt's original proofs [1984]. Instead we prefer more recent proof techniques and introduce new ones, for their ease of formalization and their robustness over different evaluation paradigms. Takahashi's trick [1994] to prove call-by-name genericity is such a technique which we build on and which led us to a reformulation of the contextual preorder without references to contexts (defining a *substitution preorder* instead).

The Call-by-Value Case. We study the more challenging case of call-by-value, where genericity had not been shown before. We stress that Plotkin's original CbV calculus [1975] is not suitable for our analysis of (in)equational theories and contextual preorders. A first symptom is that meaningless terms cannot be characterized as diverging for some strategy in Plotkin's calculus (while such characterization easily holds in CbN). We then adopt Accattoli and Paolini's *value substitution calculus* (VSC) [2012], which is an extension of Plotkin's calculus solving its well-known issues with open terms and having the same (closed) contextual equivalence. Therein, meaningless terms are characterized as those diverging for the VSC weak evaluation.

We are able to prove light genericity for the VSC. This thesis provides several proofs, all different to the CbN case, because Takahashi's technique does not easily adapt to the CbV case: a semantic proof based on Ehrhard's CbV multi types [2012]; and two proofs based on operational program equivalences, namely applicative similarity and normal form similarity.

A difference between the head CbN case and weak CbV case is given by an interesting class of terms, those evaluating to an infinite sequence of abstractions, that is, such that $t \to^*_\beta \lambda x.t'$ with t' having the same property. Such terms are head diverging (thus head CbN meaningless), but weak CbV normalizing (CbV meaningful), and hereditarily so. We prove that these *super*

(normalizing) terms are maximum elements of the CbV contextual pre-order, and the statement of this fact corresponds to the new notion of *co-genericity*.

Again, we provide different proof techniques to obtain CbV co-genericity: an operational proof based on Takahashi's technique for light genericity; and a proof based on applicative similarity. We do not use Ehrhard's CbV multi types [2012] nor normal form similarities because super terms are not maximum therein.

QUANTITATIVE REFINEMENTS OF CONTEXTUAL EQUIVALENCE (PART II)

The second part of this thesis contains an important contribution introducing *interaction equivalence*, a quantitative refinement of contextual equivalence. Let us first discuss an already known quantitative refinement, taking into account execution time.

Cost Equivalence. It is natural to refine contextual equivalence by replacing the termination predicate $\Downarrow$ with a finer one $\Downarrow^k$ indicating termination in k evaluation steps (we omit the observation $\mathtt{obs}$). Intuitively, the number of evaluation steps is taken as a time cost model. With the compiler analogy in mind, it is indeed natural to ask that a program transformation preserves the time behavior. This refinement is exactly Sands' notion of *cost equivalence*, the symmetric form of his more famous *improvement preorder* [1996a,b, 1999]:

$$t \equiv^{\mathrm{cost}} u \quad \text{if for all contexts } C, \forall k \geq 0\,. \quad [\, C\langle t\rangle \Downarrow^k \quad \Leftrightarrow \quad C\langle u\rangle \Downarrow^k \,]$$

Cost equivalence falls short, however, of being a refinement of contextual equivalence. The price to pay for the added quantitative information is that it can no longer be seen as a semantics, that is, it is *not* an equational theory. The reason is both simple and deep. It is simple because (in, say, the λ-calculus) if $t \to_\beta u$, then $C\langle u\rangle$ might take one step less to terminate than $C\langle t\rangle$, so β-conversion $=_\beta$ cannot be included in cost equivalence $\equiv^{\mathrm{cost}}$, breaking the invariance of equational theories.

Internal and External. The issue is deep as it reflects a more general tension between quantitative intensional properties such as time, which by their nature cannot be invariant under evaluation, and semantical notions such as equational theories, that are expected to hide the computational process. More generally, one might identify two perspectives on programs. The *internal view* studies programs *in isolation*, and it is concerned with qualitative properties such as, e. g., termination, confluence, productivity, or quantitative properties such as time or space cost. The *external view*, instead, studies how programs *interact* with other programs, as it is the case for contextual equivalences, labelled transition systems (LTSs), or process calculi. Usually, the external view hides the internal dynamics of programs, for instance via the invariance requirement for contextual equivalence, or via the silent τ transitions of LTSs.

The mentioned issue with cost equivalence (the fact that is not invariant, hence not an equational theory) is an instance of a more general question: is there a way of analyzing quantitative properties externally, without interference from the hidden internal dynamics? Concretely, is there a way of measuring the amount of *interactions* between a term and its context *modulo* the internal dynamics of the term and of the context?

Interaction Equivalence and the Checkers Calculus. THIS THESIS provides a positive answer. The literature on improvements has focused on application-oriented λ-calculi based on call-by-need evaluation. Our focus is different, more theoretical, we are rather interested in the semantical aspect and the understanding of the internal/external dilemma. Therefore, we place ourselves in a minimalistic setting, namely the ordinary (call-by-name) untyped λ-calculus. We introduce a new notion of cost equivalence, called *(head) interaction equivalence*, whose key

property is being—as we prove—an equational theory, in contrast to what happens for cost equivalence. This is obtained via a reconciliation of the internal and the external perspectives, loosely inspired by *game semantics* [Abramsky et al., 2000; Hyland and Ong, 2000].

Interaction equivalence is based on the new *checkers calculus* $\Lambda_{\circ\bullet}$, a λ-calculus enriched with two players, black $\bullet$ and white $\circ$. The idea is to duplicate the abstraction and application constructors of the λ-calculus as to have white ($\lambda_\circ x.t$ and $t \circ u$) and black variants ($\lambda_\bullet x.t$ and $t \bullet u$) of each. Next, borrowing from LTSs, one defines two variants of β-reduction, the silent one (internal to a player) and the interaction (external) one, depending on whether the constructors involved in the redex belong to the same player or not:

Silent β			Interaction β		
$(\lambda_\bullet x.t) \bullet u$	$\to_{\beta_\tau}$	$t\{x \leftarrow u\}$	$(\lambda_\circ x.t) \bullet u$	$\to_{\beta_\bullet}$	$t\{x \leftarrow u\}$
$(\lambda_\circ x.t) \circ u$	$\to_{\beta_\tau}$	$t\{x \leftarrow u\}$	$(\lambda_\bullet x.t) \circ u$	$\to_{\beta_\bullet}$	$t\{x \leftarrow u\}$

Interaction equivalence of t and u is then defined in the checkers calculus exactly as cost equivalence *except* that one uses the predicate $\Downarrow^{\bullet k}_{\mathrm{h}_{\circ\bullet}}$ holding when checkers head reduction terminates using k interaction steps $\to_{\beta_\bullet}$ and—crucially—*arbitrarily many* silent steps $\to_{\beta_\tau}$.

Checkers interaction equivalence is then transferred to the ordinary λ-calculus via a *paint-and-wash* construction: two ordinary λ-terms t and u are interaction equivalent when their uniformly, say, black-painting $\overline{t}^\bullet$ and $\overline{u}^\bullet$ are interaction equivalent in the checkers calculus, i. e.:

$$t \equiv^{\mathrm{int}} u \text{ if } \quad \text{for all checkers contexts } C, \text{ for all } k \geq 0, \quad [\, C\langle \overline{t}^\bullet \rangle \Downarrow^{\bullet k}_{\mathrm{h}_{\circ\bullet}} \quad \Leftrightarrow \quad C\langle \overline{u}^\bullet \rangle \Downarrow^{\bullet k}_{\mathrm{h}_{\circ\bullet}} \,]$$

Our first result about interaction equivalence is that it is an equational theory of the ordinary untyped λ-calculus. As for contextual equivalence, proving that $\equiv^{\mathrm{int}}$ is an equational theory is non-trivial, and the counting of interaction steps adds a further difficulty. We also prove that two preorder variants of interaction equivalence, one of which is the interaction reformulation of Sands's improvements, are *in*equational theories. The thesis includes rewriting proofs of these facts, which were not included in the paper [Accattoli, Lancelot, Manzonetto, et al., 2025].

Inspecting Black Boxes. Contextual equivalences, as already mentioned, are black-box principles, natural to define but hard to establish. It is then common to look for alternative, more explicit reformulations that relate the inner structure of equivalent terms. Historically, these reformulations were presented via semantic trees such as *Böhm trees*, introduced by Barendregt [1977b]. Head contextual equivalence $\equiv^{\mathrm{ctx}}_{\mathrm{h}}$ for the untyped λ-calculus is one of the few cases for which an explicit description is available, as they are not easy to obtain. Namely, $\equiv^{\mathrm{ctx}}_{\mathrm{h}}$ was characterized by Hyland [1976] and Wadsworth [1976] as the equational theory $\mathcal{B}\eta^\infty$ induced by the equality of Böhm trees up to possibly infinite η-equivalence.

In this thesis, we provide an explicit description of interaction equivalence.

Theorem (**3.5.6** at p.125 ∈ Lancelot 2025a)**.** *Interaction equivalence is exactly the equational theory $\mathcal{B}$ induced by Böhm tree equality.*

In particular, interaction equivalence is *not* extensional, that is, it does not validate any form of η-equivalence, given that x and $\lambda y.xy$ clearly have a different number of interactions with any context providing an argument. Therefore, interaction equivalence $\equiv^{\mathrm{int}}$ has a simpler explicit description than head contextual equivalence $\equiv^{\mathrm{ctx}}_{\mathrm{h}}$, and provides an operational insight about η-equivalence. The failure of η-equivalence also reveals that, despite the analogy, our framework is not that close to game semantics, where η is naturally validated unless additional constraints are imposed to avoid it, as in [Ker et al., 2003; Ong and Di Gianantonio, 2002].

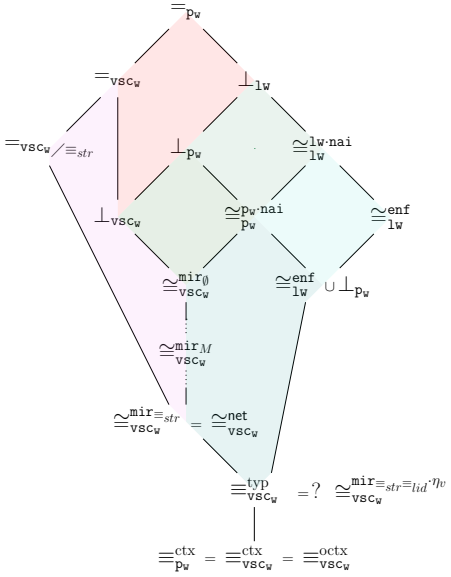
This result turns out to *solve an open problem* in the semantical theory of the untyped λ-calculus, namely the problem of finding a satisfying description of (non-extensional) Böhm tree equality $\mathcal{B}$ as an observational equivalence. Ours is the first solution that does not require adding any extra features such as concurrency or non-determinism, only refining the analysis.

CALL-BY-VALUE PROGRAM PREORDERS & EQUIVALENCES (PART III)

The third part of this thesis contains the second major contribution, which is a study of the theory of call-by-value and its program equivalences/preorders. Properties of program equivalences are brittle, as they are not preserved by extensions of the calculus under study, nor by restrictions, and not even by changing the evaluation strategy within the same calculus—which makes the switch to call-by-value non trivial.

Some notions adapt to the call-by-value case, such as contextual equivalence which is easily defined in call-by-value but its nice properties might be harder to prove (such as characterizing minimum terms and proving that contextual equivalence is maximal). This work stems from the observation that natural program equivalences, such as Sangiorgi's *normal form bisimilarity* [1994] (shortened to nf-bisimilarity), behave differently in call-by-name and call-by-value.

Normal Form Bisimilarity. Normal form bisimulations are program equivalences that, instead of comparing terms *externally*, depending on how they behave *in contexts*, compare them *internally*, by looking at the structure of their (infinitary) *normal forms*. Sangiorgi's CbN nf-bisimilarity is not fully abstract for contextual equivalence, being sound but not complete. The failure of full abstraction is compensated by the fact that CbN nf-bisimilarity is easier to establish than, say, applicative similarity, because of the absence of quantification over arguments. Typically, it is easy to show that different fix-points combinators—which are the paradigmatic terms with infinitary normal forms—are nf-bisimilar, while it is hard to show that they are applicative bisimilar.



THIS THESIS presents an array of normal form bisimulations for call-by-value evaluation, proofs that they are sound with respect to contextual equivalence and a denotational program equivalence induced by a special type system (namely non-idempotent intersection type). The *kite of call-by-value theories* (see on the right) portrays the wide range of possible program equivalences that make sense for call-by-value, with a much richer landscape than in the call-by-name theory of which we try to replicate Barendregt's kite of λ-theories [1984].

The CbV nf-bisimilarity of reference is that of Lassen's *enf bisimilarity* $\simeq^{\mathtt{enf}}_{\mathtt{lw}}$ [2005]. Like Sangiorgi's, it is sound but not complete for CbV contextual equivalence $=^{\mathtt{ctx}}_{\mathtt{pw}}$.

We introduce other call-by-value normal form (bi)similarities and study them against Lassen's enf bisimilarity. The largest nf-bisimilarity newly introduced in this thesis is *net bisimilarity*, which we call "net" because of its relationship to linear logic proof nets and the term identifications they induce.

Theorem (**7.4.8** at p.218 ∈ Lancelot 2025a)**.** *Net bisimilarity* $\simeq^{\mathtt{net}}_{\mathtt{w}}$ *is compatible, hence it is an equational theory.*

Our study is more a new beginning than the end of the story: net bisimilarity $\simeq^{\mathtt{net}}_{\mathtt{w}}$ is *not* a refinement of Lassen's $\simeq^{\mathtt{enf}}_{\mathtt{lw}}$. Indeed, the two are *incomparable*: neither net nor enf includes the other. The main examples of such equations are Moggi's laws [1988, 1989], validated only by Lassen, and the identification of meaningless terms as well as the commutation of lets, validated only by net bisimilarity.

All program equivalences we consider fail to reach call-by-value contextual equivalence, by design (normal form bisimilarities cannot reach easily full abstraction in CbV) and by choice: contextual equivalence for call-by-value behaves in a non-call-by-value manner. Indeed, we show that call-by-value contextual equivalence is somehow *blind-to-efficiency* [Accattoli and Lancelot, 2024b]: while call-by-value is designed as an efficient evaluation paradigm, its contextual equivalence accepts the naive duplications done by call-by-name evaluation.

The blindness-to-efficiency *result is only partially covered, to avoid another lengthy and technical development in the thesis.*

Impasse and Beyond. Summing up, it seems that one cannot reconcile nf-bisimilarity satisfying either commuting lets and identifying meaningless terms with those satisfying Moggi's laws, or at least it is far from evident how to do it. We explore this *impasse* in PART III: we study two further program equivalences, a naive bisimilarity $\cong^{\mathtt{nai}}_{\mathtt{p_w}}$ and the program equivalence $\simeq_{type}$ induced by Ehrhard's CbV relational model [Ehrhard, 2012] (presented in this thesis with multi types), which are sort of the intersection and the union of enf and net bisimilarities, and provide an overview of how all these equational theories relate to each other.

REFERENCES

S. Abramsky. 1990. "The Lazy Lambda Calculus." In: *Research Topics in Functional Programming*. ISBN: 0201172364.

S. Abramsky, R. Jagadeesan, and P. Malacaria. 2000. "Full Abstraction for PCF." *Inf. Comput.*, 163, 2, 409–470. DOI: 10.1006/INCO.2000.2930.

B. Accattoli and G. Guerrieri. 2022. "The theory of call-by-value solvability." *Proc. ACM Program. Lang.*, 6, ICFP, 855–885. DOI: 10.1145/3547652.

B. Accattoli and A. Lancelot. 2024a. "Light Genericity." In: *FoSSaCS 2024*. ISBN: 978-3-031-57231-9.

B. Accattoli and A. Lancelot. 2024b. "Mirroring Call-By-Need, or Values Acting Silly." In: *9th International Conference on Formal Structures for Computation and Deduction, FSCD 2024*. DOI: 10.4230/LIPICS.FSCD.2024.23.

B. Accattoli, A. Lancelot, and C. Faggian. 2023. "Normal Form Bisimulations By Value." DOI: 10.48550/ARXIV.2303.08161.

B. Accattoli, A. Lancelot, G. Manzonetto, and G. Vanoni. 2025. "Interaction Equivalence." *Proc. ACM Program. Lang.* POPL. DOI: 10.1145/3704891.

B. Accattoli and L. Paolini. 2012. "Call-by-Value Solvability, Revisited." In: *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012*. Ed. by T. Schrijvers and P. Thiemann. DOI: 10.1007/978-3-642-29822-6_4.

Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. 1995. "The Call-by-Need Lambda Calculus." In: *POPL'95*. Ed. by R. K. Cytron and P. Lee. DOI: 10.1145/199448.199507.

H. Barendregt. 1971. "Some extensional term models for combinatory logics and λ-calculi." Ph.D. Dissertation. Univ. Utrecht.

H. Barendregt. 1977a. "Solvability in lambda-calculi." In: *Colloque international de logique : Clermont-Ferrand, 18-25 juillet 1975*. Ed. by M. Guillaume.

H. Barendregt. 1977b. "The Type Free Lambda Calculus." In: *Handbook of Mathematical Logic*. Studies in Logic and the Foundations of Mathematics.

H. Barendregt. 1984. *The Lambda Calculus – Its Syntax and Semantics*. Studies in logic and the foundations of mathematics. Vol. 103. North-Holland. ISBN: 9780444867483.

H. Barendregt and G. Manzonetto. 2022. *A Lambda Calculus Satellite*. College Publications. ISBN: 978-1-84890-415-6. https://www.collegepublications.co.uk/logic/mlf/?00035.

T. Ehrhard. 2012. "Collapsing non-idempotent intersection types." In: *Computer Science Logic (CSL'12), France*. DOI: 10.4230/LIPIcs.CSL.2012.259.

M. Hyland. 1976. "A Syntactic Characterization of the Equality in Some Models for the Lambda Calculus." *Journal of the London Mathematical Society*. DOI: https://doi.org/10.1112/jlms/s2-12.3.361.

M. Hyland and C. L. Ong. 2000. "On Full Abstraction for PCF: I, II, and III." *Inf. Comput.* DOI: 10.1006/INCO.2000.2917.

A. D. Ker, H. Nickau, and C. L. Ong. 2003. "Adapting innocent game models for the Böhm tree λ-theory." *Theor. Comput. Sci.* DOI: 10.1016/S0304-3975(02)00849-6.

D. Kesner. 2016. "Reasoning About Call-by-need by Means of Types." In: *FOSSACS 2016*. DOI: 10.1007/978-3-662-49630-5_25.

A. Lancelot. Nov. 2025a. "Comparing functional programs, or how to put λ-terms back in order." Theses. Institut Polytechnique de Paris, [Nov. 2025]. https://theses.hal.science/tel-05407883.

A. Lancelot. 2025b. "Separating Terms by Means of Multi Types, Coinductively." In: *30th International Conference on Types for Proofs and Programs (TYPES 2024)*. Ed. by R. E. Møgelberg and B. van den Berg. ISBN: 978-3-95977-376-8. DOI: 10.4230/LIPIcs.TYPES.2024.4.

A. Lancelot, B. Accattoli, and M. Vemclefs. 2025. "Barendregt's Theory of the λ-Calculus, Refreshed and Formalized." In: *16th International Conference on Interactive Theorem Proving (ITP 2025)*. Abella source code available at https://github.com/adrilancelot/Abella-lambda-Barendregt-theory. ISBN: 978-3-95977-396-6. DOI: 10.4230/LIPIcs.ITP.2025.13.

S. Lassen. 2005. "Eager Normal Form Bisimulation." In: [LICS '05]. ISBN: 0769522661. DOI: 10.1109/LICS.2005.15.

E. Moggi. 1988. *Computational λ-Calculus and Monads*. LFCS report ECS-LFCS-88-66. University of Edinburgh.

E. Moggi. 1989. "Computational λ-Calculus and Monads." In: *LICS '89*. DOI: 10.1109/LICS.1989.39155.

J. H. Morris. 1968. "Lambda-calculus Models of Programming Languages." Ph.D. Dissertation. Massachusetts Institute of Technology. https://books.google.is/books?id=DklAAQAAIAAJ.

C. L. Ong and P. Di Gianantonio. 2002. "Games Characterizing Lévy-Longo Trees." In: *ICALP 2002*. Vol. 2380. Springer, 476–487. DOI: 10.1007/3-540-45465-9_41.

M. Patrignani, A. Ahmed, and D. Clarke. 2019. "Formal Approaches to Secure Compilation: A Survey of Fully Abstract Compilation and Related Work." *ACM Comput. Surv.* DOI: 10.1145/3280984.

G. D. Plotkin. 1975. "Call-by-name, call-by-value and the λ-calculus." *Theoretical Computer Science*, 1, 2, 125–159. DOI: https://doi.org/10.1016/0304-3975(75)90017-1.

D. Sands. 1996a. "Proving the Correctness of Recursion-Based Automatic Program Transformations." *Theor. Comput. Sci.* DOI: 10.1016/0304-3975(96)00074-6.

D. Sands. 1996b. "Total correctness by local improvement in the transformation of functional programs." *ACM Trans. Program. Lang. Syst.* DOI: 10.1145/227699.227716.

D. Sands. 1999. "Improvement theory and its applications." In: *Higher Order Operational Techniques in Semantics*. Cambridge University Press, USA. ISBN: 0521631688.

D. Sangiorgi. 1994. "The Lazy Lambda Calculus in a Concurrency Scenario." *Information and Computation*, 111, 1, 120–153. DOI: https://doi.org/10.1006/inco.1994.1042.

D. S. Scott. 1972. "Continuous lattices." In: *Toposes, Algebraic Geometry and Logic*. DOI: 10.1007/BFb0073967.

M. Takahashi. 1994. "A simple proof of the genericity lemma." In: *Logic, Language and Computation: Festschrift in Honor of Satoru Takasu*. DOI: 10.1007/BFb0032397.

C. P. Wadsworth. 1971. "Semantics and pragmatics of the lambda-calculus." PhD Thesis. University of Oxford.

C. P. Wadsworth. 1976. "The Relation Between Computational and Denotational Properties for Scott's $\mathcal{D}_\infty$-Models of the Lambda-Calculus." *SIAM J. Comput.* DOI: 10.1137/0205036.