Tesis de Licenciatura

Título

Gonzalo Ciruelos Rodríguez

Director: Pablo Barenbaum

Fecha de Presentación: ???

Contents

En sistemas de tipos intersección no idempotentes típicos, la normalización de pruebas no es confluente. En este trabajo presentamos un sistema confluente de tipos intersección no idempotentes para el cálculo λ. Escribimos las derivaciones de tipos usando una sintáxis concisa de términos de prueba. El sistema goza de buenas propiedades: subject reduction, es fuertemente normalizante, y tiene una teoría de residuos muy regular. Establecemos una correspondencia con el cálculo lambda mediante teoremas de simulación.

La maquinaria de los tipos intersección no idempotentes nos permite seguir el rastro del uso de los recursos necesarios para obtener una respuesta. En particular, induce una noción de *basura*: un cómputo es basura si no contribuye a hallar una respuesta. Usando estas nociones, mostramos que el espacio de derivaciones de un término λ puede ser factorizado usando una variante de la construcción de Grothendieck para semireticulados. Esto significa, en particular, que cualquier derivación del cálculo λ puede ser escrita de una única manera como un prefijo libre de basura, seguido de basura.

Palabras clave: Cálculo lambda, Tipos intersección, Espacio de derivacion, Reticulado

In typical non-idempotent intersection type systems, proof normalization is not confluent. In this work we introduce a confluent non-idempotent intersection type system for the λ-calculus. Typing derivations are presented using a concise proof term syntax. The system enjoys good properties: subject reduction, strong normalization, and a very regular theory of residuals. A correspondence with the λ-calculus is established by simulation theorems.

The machinery of non-idempotent intersection types allows us to track the usage of resources required to obtain an answer. In particular, it induces a notion of *garbage*: a computation is garbage if it does not contribute to obtaining an answer. Using these notions, we show that the derivation space of a λ-term may be factorized using a variant of the Grothendieck construction for semilattices. This means, in particular, that any derivation in the λ-calculus can be uniquely written as a garbage-free prefix followed by garbage.

Keywords: Lambda Calculus, Intersection Types, Derivation Space, Lattice
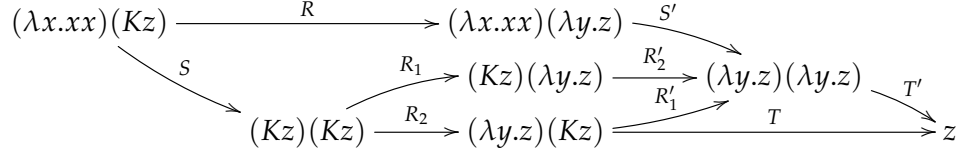
Introduction

The space of computations of a program may have a complex structure. Consider a side-effect free programming language. The possible computations of a tuple (A, B) are rewrite sequences $(A, B) \rightarrow \ldots \rightarrow (A', B')$. These sequences can always be decomposed as two non-interfering computations $A \rightarrow \ldots \rightarrow A'$ and $B \rightarrow \ldots \rightarrow B'$. The reason is that the subexpressions A and B cannot interact with each other. Indeed, the space of computations of (A, B) can be understood as the product of the spaces of A and B. In contrast, the space of computations of a function application $f(A)$ is not so easy to characterize. The difficulty is that f may use the value of A zero, one, or possibly many times.

The motivation of this paper is attempting to **understand spaces of computations**. We hope that this may be helpful to better understand the properties of evaluation strategies, such as call-by-name or call-by-value, from a quantitative point of view. A better understanding may also suggest program optimizations, and it should allow to justify that certain program conversions are sound: e.g. that they do not turn a terminating program into a non-terminating one.

The pure λ-calculus is the quintessential functional programming language. Computations in the λ-calculus have been thoroughly studied since its conception in the 1930s. The well-known theorem by Church and Rosser [CR36] states that the λ-calculus is confluent, which means, in particular, that terminating programs have unique normal forms. Another result by Curry and Feys [CF58] states that computations in the λ-calculus may be standardized, meaning that they may be converted into a computation in canonical form. A refinement of this theorem by Lévy [Lév78] asserts that the canonical computation thus obtained is equivalent to the original one in a strong sense, namely that they are permutation equivalent. In a series of papers [Mel97, Mel00, Mel02a, Mel02b, Mel05], Melliès generalized many of these results to the abstract setting of axiomatic rewrite systems.

Let us discuss "spaces of computations" more precisely. The derivation space of an object x in some rewriting system is the set of all derivations, i.e. sequences of rewrite steps, starting from x. In this paper, our rewriting system of interest will be the pure λ-calculus, and we will be interested in finite derivations only. In the λ-calculus, a transitive relation between derivations may be defined, the prefix order. A derivation ρ is a

prefix of a derivation σ, written $\rho \sqsubseteq \sigma$, whenever ρ performs less computational work than σ. Formally, $\rho \sqsubseteq \sigma$ is defined to hold whenever the projection ρ/σ is empty[1]. For example, if $K = \lambda x.\lambda y.x$, the derivation space of the term $(\lambda x.xx)(Kz)$ can be depicted with the reduction graph below. Derivations are directed paths in the reduction graph, and ρ is a prefix of σ if there is a directed path from the target of ρ to the target of σ. For instance, SR_2 is a prefix of $RS'T'$:

$$
\begin{array}{l}
(\lambda x.xx)(Kz) \xrightarrow{\quad R \quad} (\lambda x.xx)(\lambda y.z) \xrightarrow{S'} \\
\qquad \searrow S \qquad\qquad \xrightarrow{R_1} (Kz)(\lambda y.z) \xrightarrow[R_1']{R_2'} (\lambda y.z)(\lambda y.z) \searrow T' \\
\qquad\qquad (Kz)(Kz) \xrightarrow{R_2} (\lambda y.z)(Kz) \xrightarrow{\quad T \quad} z
\end{array}
$$

Remark that the relation $\sqsubseteq$ is reflexive and transitive but not antisymmetric, i.e. it is a quasi-order but not an order. For example $RS' \sqsubseteq SR_1R_2' \sqsubseteq RS'$ but $RS' \neq SR_1R_2'$. Antisymmetry may be recovered as usual when in presence of a quasi-order, namely by working modulo permutation equivalence: two derivations ρ and σ are said to be permutation equivalent, written $\rho \equiv \sigma$, if $\rho \sqsubseteq \sigma$ and $\sigma \sqsubseteq \rho$. Working modulo permutation equivalence is reasonable because Lévy's formulation of the standardization theorem ensures that permutation equivalence is decidable, and each equivalence class has a canonical representative.
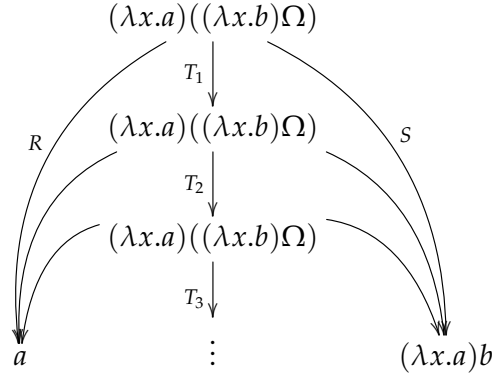
Derivation spaces are known to exhibit various regularities [Lév78, Zil84, Lan94, Mel96, Lev15, AL13]. In his PhD thesis, Lévy [Lév78] showed that the derivation space of a term is an upper semilattice: any two derivations ρ, σ from a term t have a least upper bound $\rho \sqcup \sigma$, defined as $\rho(\sigma/\rho)$, unique up to permutation equivalence. On the other hand, the derivation space of a term t is not an easy structure to understand in general[2]. For example, relating the derivation space of an application ts with the derivation spaces of t and s appears to be a hard problem. Lévy also noted that the greatest lower bound of two derivations does not necessarily exist, meaning that the derivation space of a term does not form a lattice in general. Even when it forms a lattice, it may not necessarily be a distributive lattice, as observed for example by Laneve [Lan94].

Consider the following counterexample, due to Lévy, showing that the meet of derivations is not well-defined. Let $\Omega = (\lambda x.xx)\lambda x.xx$, and consider the reduction

---

[1]The notion of projection defined by means of residuals is the standard one, see e.g. [Bar84, Chapter 12] or [Ter03, Section 8.7].

[2]Problem 2 in the RTA List of Open Problems [DJK91] poses the open-ended question of investigating the properties of "spectra", i.e. derivation spaces.

space of $(\lambda x.a)((\lambda x.b)\Omega)$, where the steps T_i contract Ω:

$$
\begin{array}{c}
(\lambda x.a)((\lambda x.b)\Omega) \\
\Big\downarrow T_1 \\
(\lambda x.a)((\lambda x.b)\Omega) \\
\Big\downarrow T_2 \\
(\lambda x.a)((\lambda x.b)\Omega) \\
\Big\downarrow T_3 \\
a \qquad \vdots \qquad (\lambda x.a)b
\end{array}
$$

Observe that for all $n \in \mathbb{N}$ we have that $T_1 \ldots T_n \sqsubseteq R$ and $T_1 \ldots T_n \sqsubseteq S$ so the meet $R \sqcap S$ does not exist.

In [Mel97], Melliès showed that derivation spaces in any rewriting system satisfying certain axioms may be factorized using two spaces, one of <u>external</u> and one of <u>internal</u> derivations.

The difficulty to understand derivation spaces is due to three pervasive phenomena of <u>interaction</u> between computations. The first phenomenon is <u>duplication</u>: in the reduction graph of above, the step S duplicates the step R, resulting in two copies of R: the steps R_1 and R_2. In such situation, one says that R_1 and R_2 are <u>residuals</u> of R, and, conversely, R is an <u>ancestor</u> of R_1 and R_2. The second phenomenon is <u>erasure</u>: in the diagram above, the step T erases the step R_1', resulting in no copies of R_1'. The third phenomenon is <u>creation</u>: in the diagram above, the step R_2 creates the step T, meaning that T is not a residual of a step that existed prior to executing R_2; that is, T has no ancestor.

These three interaction phenomena, especially duplication and erasure, are intimately related with the management of <u>resources</u>. In this work, we aim to explore the hypothesis that **having an explicit representation of resource management may provide insight on the structure of derivation spaces**.

There are many existing λ-calculi that deal with resource management explicitly [Bou93, ER03, KL07, KR], most of which draw inspiration from Girard's Linear Logic [Gir87]. In recent years, one family of such formalisms, namely calculi endowed with <u>non-idempotent intersection type systems</u>, has received some attention [Ehr12, BL13, BKDR14, BKV17, Kes16, Via17, KRV18]. These type systems are able to statically capture non-trivial dynamic properties of terms, particularly <u>normalization</u>, while at the same time being amenable to elementary proof techniques by induction, rather than arguments based on reducibility. Intersection types were originally proposed by Coppo and Dezani-Ciancaglini [CD78] to study termination in the λ-calculus. They are characterized by the presence of an <u>intersection</u> type constructor $\tau \cap \sigma$. <u>Non-idempotent</u> intersection type systems are distinguished from their usual idempotent counterparts by the fact

that intersection is not declared to be idempotent, i.e. τ and $\tau \cap \tau$ are not equivalent types. Rather, intersection behaves like a multiplicative connective in linear logic. Arguments to functions are typed many times, typically once per each time that the argument will be used. Non-idempotent intersection types were originally formulated by Gardner [Gar94], and later reintroduced by de Carvalho [Car07].

In this work, we will use a non-idempotent intersection type system based on system $\mathcal{W}$ of [BKV17] (called system $\mathcal{H}$ in [BKDR14]). Let us recall its definition. Terms are as usual in the λ-calculus ($t ::= x \mid \lambda x.t \mid t\,t$). Types $\tau, \sigma, \rho, \ldots$ are defined by the grammar:

$$\tau \quad ::= \quad \alpha \mid \mathcal{M} \to \tau \qquad\qquad \mathcal{M} \quad ::= \quad [\tau_i]_{i=1}^n \quad \text{with } n \geq 0$$

where α ranges over one of denumerably many base types, and $\mathcal{M}$ represents a multiset of types. Here $[\tau_i]_{i=1}^n$ denotes the multiset containing the types $\tau_1, \ldots, \tau_n$ with their respective multiplicities. A multiset $[\tau_i]_{i=1}^n$ intuitively stands for the (non-idempotent) intersection $\tau_1 \cap \ldots \cap \tau_n$. The sum of multisets $\mathcal{M} + \mathcal{N}$ is defined as their union (adding multiplicities). A typing context Γ is a partial function mapping variables to multisets of types. The domain of Γ is the set of variables x such that $\Gamma(x)$ is defined. We assume that typing contexts always have finite domain and hence they may be written as $x_1 : \mathcal{M}_1, \ldots, x_n : \mathcal{M}_n$. The sum of contexts $\Gamma + \Delta$ is their pointwise sum, i.e. $(\Gamma + \Delta)(x) := \Gamma(x) + \Delta(x)$ if $\Gamma(x)$ and $\Delta(x)$ are both defined, $(\Gamma + \Delta)(x) := \Gamma(x)$ if $\Delta(x)$ is undefined, and $(\Gamma + \Delta)(x) := \Delta(x)$ if $\Gamma(x)$ is undefined. We write $\Gamma +_{i=1}^n \Delta_i$ to abbreviate $\Gamma + \Delta_1 + \ldots + \Delta_n$. The disjoint sum of contexts $\Gamma \oplus \Delta$ stands for $\Gamma + \Delta$, provided that the domains of Γ and Δ are disjoint. A typing judgment is a triple $\Gamma \vdash t : \tau$, representing the knowledge that the term t has type τ in the context Γ. Type assignment rules for system $\mathcal{W}$ are as follows.

Definition 1 (System $\mathcal{W}$).

$$\frac{}{x : [\tau] \vdash \tau}\ \text{var} \qquad \frac{\Gamma \oplus (x : \mathcal{M}) \vdash t : \tau}{\Gamma \vdash \lambda x.t : \mathcal{M} \to \tau}\ \to_{\text{I}} \qquad \frac{\Gamma \vdash t : [\sigma_i]_{i=1}^n \to \tau \quad (\Delta_i \vdash s_i : \sigma_i)_{i=1}^n}{\Gamma +_{i=1}^n \Delta_i \vdash t\,s : \tau}\ \to_{\text{E}}$$

Observe that the $\to_{\text{E}}$ rule has $n + 1$ premises, where $n \geq 0$. System $\mathcal{W}$ enjoys various properties, nicely summarized in [BKV17].

There are two obstacles to adopting system $\mathcal{W}$ for studying derivation spaces. The first obstacle is just a matter of presentation—typing derivations use a tree-like notation, which is cumbersome. One would like to have an alternative presentation based on proof terms. For example, one would like to write x^τ for an application of the var rule, $\lambda x.t$ for an application of the $\to_{\text{I}}$ rule, and $t[s_1, \ldots, s_n]$ for an application of the $\to_{\text{E}}$ rule, so that, for example, $\lambda x.x^{[\alpha,\alpha] \to \beta}[x^\alpha, x^\alpha]$ represents the following typing derivation:

$$\frac{\dfrac{}{x : [\alpha, \alpha] \to \beta \vdash x : [\alpha, \alpha] \to \beta}\ \text{var} \quad \dfrac{}{x : [\alpha] \vdash x : \alpha}\ \text{var} \quad \dfrac{}{x : [\alpha] \vdash x : \alpha}\ \text{var}}{\dfrac{x : [[\alpha, \alpha] \to \beta, \alpha, \alpha] \vdash xx : \beta}{\vdash \lambda x.xx : [[\alpha, \alpha] \to \beta, \alpha, \alpha] \to \beta}\ \to_{\text{I}}}\ \to_{\text{E}}$$

The second obstacle is a major one for our purposes: <u>proof normalization</u> in this system is not confluent. The reason is that applications take multiple arguments, and a β-reduction step must choose a way to distribute these arguments among the occurrences of the formal parameters. For instance, the following critical pair cannot be closed:

$$(\lambda x. y^{[\alpha]\to[\alpha]\to\beta}[x^\alpha][x^\alpha])[z^{[\gamma]\to\alpha}[z^\gamma], z^{[]\to\alpha}[]] \longrightarrow y^{[\alpha]\to[\alpha]\to\beta}[z^{[\gamma]\to\alpha}[z^\gamma]][z^{[]\to\alpha}[]]$$
$$y^{[\alpha]\to[\alpha]\to\beta}[z^{[]\to\alpha}[]][z^{[\gamma]\to\alpha}[z^\gamma]]$$

The remainder of this work is organized as follows:

- In Chapter 1, we review some standard notions of order and rewriting theory, as well as some basic notions of the λ-calculus that we will use throughout the work.

- In Chapter 2, we introduce a confluent calculus $\lambda^\#$ based on system $\mathcal{W}$. The desirable properties of system $\mathcal{W}$ of [BKV17] still hold in $\lambda^\#$. Moreover, $\lambda^\#$ is confluent. We impose confluence forcibly, by decorating subtrees with distinct labels, so that a β-reduction step may distribute the arguments in a unique way. Derivation spaces in $\lambda^\#$ have very regular structure, namely they are distributive lattices.

- In Section 2.7, we establish a correspondence between derivation spaces in the λ-calculus and the $\lambda^\#$-calculus via simulation theorems, which defines a morphism of upper semilattices.

- In Chapter 3, we introduce the notion of a garbage derivation. Roughly, a derivation in the λ-calculus is <u>garbage</u> if it maps to an empty derivation in the $\lambda^\#$-calculus. This gives rise to an orthogonal notion of <u>garbage-free</u> derivation. The notion of garbage-free derivation is closely related with the notions of <u>needed step</u> [Ter03, Section 8.6], <u>typed occurrence of a redex</u> [BKV17], and <u>external</u> derivation [Mel97]. Using this notion of garbage we prove a <u>factorization theorem</u> reminiscent of Melliès' [Mel97]. The upper semilattice of derivations of a term in the λ-calculus is factorized using a variant of the Grothendieck construction. Every derivation is uniquely decomposed as a garbage-free prefix followed by a garbage suffix.

- In Conclusions, we end with a discussion of our results.

Note. Proofs including a ♠ symbol are spelled out in detail in the appendix.

Chapter 1

Preliminaries

1.1 Order theory

We are interested in understanding the derivation spaces of λ-terms. These derivation spaces, as we briefly mentioned earlier, have an structure that can be seen as an order. More specifically, the poset we will consider will be the one of derivations, where the order is giving by the amount of *work* each derivation does.

But the structures we will work with are more than just posets, they have a richer structure, some of which we will define now.

An **upper semilattice** is a poset (i.e., a set A with an order $\leq$) with a least element or **bottom** $\perp \in A$, such that for every two elements $a, b \in A$ there is a least upper bound or **join** $(a \vee b) \in A$.

A **lattice** is an upper semilattice with a greatest element or **top** $\top \in A$, and such that for every two elements $a, b \in A$ there is a greatest lower bound or **meet** $(a \wedge b) \in A$. A lattice is **distributive** if $\wedge$ distributes over $\vee$ and vice versa.

In the introduction we claimed that derivation spaces of λ-terms where upper semilattices, but in general were not lattices.

A **morphism** of upper semilattices is given by a monotonic function $f : A \to B$, i.e. $a \leq b$ implies $f(a) \leq f(b)$, preserving the bottom element, i.e. $f(\perp) = \perp$, and joins, i.e. $f(a \vee b) = f(a) \vee f(b)$ for all $a, b \in A$. Similarly for morphisms of lattices (and distributive lattices).

Any poset $(A, \leq)$ may be regarded as category whose objects are the elements of A and morphisms are of the form $b//a$ for all $a \leq b$. The category of posets with monotonic functions is denoted by Poset. In fact, we view it as a 2-category: given morphisms $f, g : A \to B$ of posets, there is a *2-cell* $f \leq g$ if $f(a) \leq g(a)$ for all $a \in A$. (A 2-category is just a category with morphisms between morphisms). This notion is more technical and will only be used in the last chapter.

1.2 Rewriting theory

The λ-calculus is a particular case of a more general mathematical concept, called **rewriting system**. Informally, a rewriting system is a set of objects and a set of rules that let you transform some objects into others.

More formally, an **axiomatic rewrite system** (cf. [Mel96, Def. 2.1]) is given by a set of objects Obj, a set of steps Stp, two functions $src, tgt : Stp \to Obj$ indicating the source and target of each step, and a **residual function** $(/)$ such that given any two steps $R, S \in Stp$ with the same source, yields a set of steps R/S such that $src(R') = tgt(S)$ for all $R' \in R/S$. Steps are ranged over by $R, S, T, \ldots$. A step $R' \in R/S$ is called a **residual** of R after S, and R is called an **ancestor** of R'. Steps are **coinitial** (resp. **cofinal**) if they have the same source (resp. target). A **derivation** is a possibly empty sequence of composable steps $R_1 \ldots R_n$. Derivations are ranged over by $\rho, \sigma, \tau, \ldots$. The functions src and tgt are extended to derivations (noting that there is a different empty derivation for each element in Obj).

Composition of derivations is defined when $tgt(\rho) = src(\sigma)$ and written $\rho\sigma$. Residuals are extended for projecting after a derivation, namely $R_n \in R_0/S_1 \ldots S_n$ if and only if there exist $R_1, \ldots, R_{n-1}$ such that $R_{i+1} \in R_i/S_{i+1}$ for all $0 \leq i \leq n-1$. Let $\mathcal{M}$ be a set of coinitial steps.

A **development** of $\mathcal{M}$ is a (possibly infinite) derivation $R_1 \ldots R_n \ldots$ such that for every index i there exists a step $S \in \mathcal{M}$ such that $R_i \in S/R_1 \ldots R_{i-1}$. A development is **complete** if it is maximal.

The definition of an abstract rewriting system is very general, and because of that it does not give us general properties for systems with such structure. In his PhD thesis, Melliès gave a set of sufficient properties (which he called axioms) that a rewriting system should have to behave properly.

An **orthogonal** axiomatic rewrite system (cf. [Mel96, Sec. 2.3]) has four additional axioms[1]:

1. Autoerasure. $R/R = \varnothing$ for all $R \in Stp$.

2. Finite Residuals. The set R/S is finite for all coinitial $R, S \in Stp$.

3. Finite Developments. If $\mathcal{M}$ is a set of coinitial steps, all developments of $\mathcal{M}$ are finite.

4. Semantic Orthogonality. Let $R, S \in Stp$ be coinitial steps. Then there exist a complete development ρ of R/S and a complete development σ of S/R such that ρ and σ are cofinal. Moreover, for every step $T \in Stp$ such that T is coinitial to R, the following equality between sets holds: $T/R\sigma = T/S\rho$.

---

[1]In [Mel96], Autoerasure is called Axiom A, Finite Residuals is called Axiom B, and Semantic Orthogonality is called PERM. We follow the nomenclature of [ABKL14]

In [Mel96], Melliès develops the theory of orthogonal axiomatic rewrite systems. A notion of **projection** ρ/σ may be defined between coinitial derivations, essentially by setting $\epsilon/\sigma \overset{\text{def}}{=} \epsilon$ and $R\rho'/\sigma \overset{\text{def}}{=} (R/\sigma)(\rho'/(\sigma/R))$ where, by abuse of notation, R/σ stands for a (canonical) complete development of the set R/σ. Using this notion, one may define a transitive relation of **prefix** ($\rho \sqsubseteq \sigma$), a **permutation equivalence** relation ($\rho \equiv \sigma$), and the **join** of derivations ($\rho \sqcup \sigma$). Some of their properties are summed up in the figure below:

Summary of properties of orthogonal axiomatic rewrite systems

$\epsilon\rho = \rho$	$\rho \sqsubseteq \sigma \overset{\text{def}}{\iff} \rho/\sigma = \epsilon$	$\rho \sqsubseteq \sigma \implies \rho/\tau \sqsubseteq \sigma/\tau$
$\rho\epsilon = \rho$	$\rho \equiv \sigma \overset{\text{def}}{\iff} \rho \sqsubseteq \sigma \wedge \sigma \sqsubseteq \rho$	$\rho \sqsubseteq \sigma \iff \tau\rho \sqsubseteq \tau\sigma$
$\epsilon/\rho = \epsilon$	$\rho \sqcup \sigma \overset{\text{def}}{=} \rho(\sigma/\rho)$	$\rho \sqcup \sigma \equiv \sigma \sqcup \rho$
$\rho/\epsilon = \rho$	$\rho \equiv \sigma \implies \tau/\rho = \tau/\sigma$	$(\rho \sqcup \sigma) \sqcup \tau = \rho \sqcup (\sigma \sqcup \tau)$
$\rho/\sigma\tau = (\rho/\sigma)/\tau$	$\rho \sqsubseteq \sigma \iff \exists\tau.\, \rho\tau \equiv \sigma$	$\rho \sqsubseteq \rho \sqcup \sigma$
$\rho\sigma/\tau = (\rho/\tau)(\sigma/(\tau/\rho))$	$\rho \sqsubseteq \sigma \iff \rho \sqcup \sigma \equiv \sigma$	$(\rho \sqcup \sigma)/\tau = (\rho/\tau) \sqcup (\sigma/\tau)$
$\rho/\rho = \epsilon$		

Let $[\rho] = \{\sigma \mid \rho \equiv \sigma\}$ denote the permutation equivalence class of ρ. In an orthogonal axiomatic rewrite system, the set $\mathbb{D}(x) = \{[\rho] \mid \mathsf{src}(\rho) = x\}$ forms an upper semilattice. The order $[\rho] \sqsubseteq [\sigma]$ is given by $\rho \sqsubseteq \sigma$, the join is $[\rho] \sqcup [\sigma] = [\rho \sqcup \sigma]$, and the bottom is $\bot = [\epsilon]$. The λ-calculus is an example of an orthogonal axiomatic rewrite system. Our structures of interest are the semilattices of derivations of the λ-calculus, written $\mathbb{D}^\lambda(t)$ for any given λ-term t. As usual, β-reduction in the λ-calculus is written $t \to_\beta s$ and defined by the contextual closure of the axiom $(\lambda x.t)s \to_\beta t\{x := s\}$.

1.3 Lists and Sets

Throughout this work we will use lists and sets, so we establish here basic definitions and notations.

Definition 2 (Lists and sets). If A is a sort, we write $\vec{A}$ for the sort of (finite) **lists** over A, defined inductively as:
$$\vec{A} ::= \epsilon \mid A \cdot \vec{A}$$

We usually write $[a_1, \ldots, a_n]$, abbreviated $[a_i]_{i=1}^n$, to stand for $a_1 \cdot (a_2 \cdot \ldots (a_n \cdot \epsilon))$. If $\vec{a}$ and $\vec{b}$ are lists, $\vec{a} + \vec{b}$ stands for its concatenation, and $|\vec{a}|$ is the length of the list $\vec{a}$. If $a, b, c, \ldots$ are the names of the metavariables ranging over a sort A, then $\vec{a}, \vec{b}, \vec{c}, \ldots$ are the names of the metavariables ranging over lists of A. When there is no possibility of confusion, we may also write $[\vec{a}, b, \vec{c}]$ for the list $\vec{a} + [b] + \vec{c}$. We write $\vec{a} \approx \vec{b}$ if $\vec{a}$ is a permutation of $\vec{b}$. Observe that $\approx$ is an equivalence relation.

In some cases we will work with (finite) **sets**, which are defined to be lists, considered modulo arbitrary permutations of their elements and without regard of the number of repetitions of each element. The notation for operations on lists will be lifted

to operations on sets. In particular, $\vec{a} + \vec{b}$ denotes the union of sets, and $|\vec{a}|$ stands for the cardinal of $\vec{a}$. This notation is chosen to resemble the multiset notation of existing intersection type systems. Whether we are referring to sets or lists will be clear from the context.

Chapter 2

A distributive λ-calculus

In this chapter we present a <u>distributive λ-calculus</u> ($\lambda^{\#}$), and we prove some basic properties it enjoys.

Terms of the $\lambda^{\#}$-calculus are typing derivations of a non-idempotent intersection type system, written using proof term syntax. The underlying type system is a variant of system $\mathcal{W}$ of [BKDR14, BKV17], the main difference being that $\lambda^{\#}$ uses <u>labels</u> and a suitable invariant on terms, to ensure that the formal parameters of all functions are in 1–1 correspondence with the actual arguments that they receive.

2.1 Types

We will now present the type system we will work with, which as we said, is a variant of the system presented in [BKV17].

To make the notation easier, we will write $\tau_1 \cap ... \cap \tau_n$ as the multiset $[\tau_1, \ldots, \tau_n]$. Recall that we work in a non idempotent environment so we may have repetitions.

Definition 3 (Types and typing contexts)**.** Let $\mathscr{E} = \{e, e', e'', \ldots\}$ be a denumerable set of labels. The sets of **types**, ranged over by $\tau, \sigma, \rho, \ldots$, and **finite sets of types**, ranged over by $\mathcal{M}, \mathcal{N}, \mathcal{P}, \ldots$, are given mutually inductively by the following abstract syntax:

$$\tau ::= \alpha^e \mid \mathcal{M} \xrightarrow{e} \tau$$

$$\mathcal{M} ::= [\tau_i]_{i=1}^n \quad \text{for some } n \geq 0$$

In a type like α^e and $\mathcal{M} \xrightarrow{e} \tau$, the label e is called the **external label**. **Typing contexts**, or contexts for short, ranged over by $\Gamma, \Delta, \Theta, \ldots$ are (total) functions from variables to finite sets of types. We write $\operatorname{dom}\Gamma$ for the set of variables x such that $\Gamma(x) \neq []$. We write $\varnothing$ for the context such that $\varnothing(x) = []$ for every variable x. The notation $\Gamma + \Delta$ stands for the **sum of contexts**, defined as follows:

$$(\Gamma + \Delta)(x) \stackrel{\text{def}}{=} \Gamma(x) + \Delta(x)$$

5

The notation $\Gamma \oplus \Delta$ stands for the **disjoint sum of contexts**, i.e. it stands for $\Gamma + \Delta$ provided $\text{dom}\,\Gamma \cap \text{dom}\,\Delta = \varnothing$. We also write $\Gamma +_{i=1}^{n} \Delta_i$ for $\Gamma + \sum_{i=1}^{n} \Delta_i$. Moreover, $x : \mathcal{M}$ denotes the context such that $(x : \mathcal{M})(x) = \mathcal{M}$ and $\text{dom}(x : \mathcal{M}) = \{x\}$.

Remark that the (only) difference with the system $\mathcal{W}$ at type-level is that we include labels.

Like in system $\mathcal{W}$, note that it will not be possible for a term to have multiple types, like the name *intersection type system* would suggest. Rather, what happens is that function terms will receive a parameter that can be interpreted as having several types.

Later we will need to look closely at these types, for which purpose the following definition will be useful.

Definition 4. A type τ is said to **occur** in another type σ if $\tau \preceq \sigma$ holds, where $(\preceq)$ is the reflexive and transitive closure of the axioms $\tau_i \preceq [\tau_1, \ldots, \tau_n] \to \sigma$ for all $i = 1..n$, and $\sigma \preceq [\tau_1, \ldots, \tau_n] \to \sigma$. This is extended to say that a type τ occurs in a multiset $[\sigma_1, \ldots, \sigma_n]$, defined by $\tau \preceq [\sigma_1, \ldots, \sigma_n]$ if $\tau \preceq \sigma_i$ for some $i = 1..n$, and that a type τ occurs in a typing context Γ, defined by $\tau \preceq \Gamma$ if $\tau \preceq \Gamma(x)$ for some $x \in \text{dom}\,\Gamma$.

2.2   Syntax

Generally, intersection type systems are used on the pure λ-calculus. In such environment, a term may be able to be typed: for example, in system $\mathcal{W}$, $\lambda x.xx$ can have the type $((\alpha \to \beta) \cap \alpha) \to \beta$. On the other hand, the term $\Omega = (\lambda x.xx)(\lambda x.xx)$ cannot be typed.

In contrast, what we will do in this work is define a slightly different calculus, such that all terms of that calculus can be typed using the non idempotent intersection type system defined above.

The informal idea behind the definition of the terms of the calculus is that in an application, for each type in the domain of the function there will be a different argument. So if we have a term t with type $(\tau_1 \cap \tau_2) \to \tau_3$, we can apply it to a *list* of arguments, one with type τ_1 and the other with type τ_2.

Definition 5 (Distributive type system)**.** The set of **distributive terms**, ranged over by $(t, s, u, \ldots)$ is given by the following abstract syntax:

$$t ::= x^\tau \mid \lambda^e x.t \mid t\,\vec{t}$$

Typing rules are defined inductively as follows.

$$\frac{}{x : [\tau] \vdash x^\tau : \tau} \text{ var} \qquad \frac{\Gamma \oplus x : \mathcal{M} \vdash t : \sigma}{\Gamma \vdash \lambda^e x.t : \mathcal{M} \xrightarrow{e} \sigma} \to_{\text{I}}$$

$$\frac{\Gamma \vdash t : [\sigma_1, \ldots, \sigma_n] \xrightarrow{e} \tau \qquad (\Delta_i \vdash s_i : \sigma_i)_{i=1}^{n}}{\Gamma +_{i=1}^{n} \Delta_i \vdash t[s_1, \ldots, s_n] : \tau} \to_{\text{E}}$$

Moreover, we introduce a judgment of the form $[\Gamma_1, \ldots, \Gamma_n] \vdash [t_1, \ldots, t_n] : [\tau_1, \ldots, \tau_n]$ with the following rule:

$$\frac{\Gamma_i \vdash t_i : \tau_i \text{ for all } i = 1..n}{[\Gamma_1, \ldots, \Gamma_n] \vdash [t_1, \ldots, t_n] : [\tau_1, \ldots, \tau_n]} \ \texttt{t-multi}$$

For example, using integer labels, $\vdash \lambda^1 x.x^{[\alpha^2, \alpha^3] \xrightarrow{4} \beta^5}[x^{\alpha^3}, x^{\alpha^2}] : [[\alpha^2, \alpha^3] \xrightarrow{4} \beta^5, \alpha^2, \alpha^3] \xrightarrow{1} \beta^5$ is a derivable judgment. For another example, $x : [] \xrightarrow{1} \alpha^2 \vdash x^{[] \xrightarrow{1} \alpha^2}[] : \alpha^2$ is a derivable judgment.

Note that writing all labels is rather cumbersome, so we will omit or simplify them when possible.

2.2.1 Correctness

Observe that the definition we gave has a fatal flaw: we cannot uniquely associate arguments with variables in the body of the lambdas. For example, consider the term $(\lambda^1 x.y^{[\alpha^2, \alpha^2] \xrightarrow{3} \alpha^4}[x^{\alpha^2}, x^{\alpha^2}])[a^{\alpha^2}, b^{\alpha^2}]$. We don't know which parameter to associate which eaich x—which parameter goes in the first x, a or b?

To solve that problem we introduce an invariant that will ensure that problem does not manifest. We will call that invariant **correctness**, and we will consider $\lambda^{\#}$ to be the system of all correct terms.

Note that the problem is that the function in the application expects two arguments with exactly the same type. A related problem is that, in the body of the function, the variable x has the same type twice (remember that in a non idempotent context repetition matters). In fact, it is enough to ask that those two anomalies don't show up for the system to work.

We will also ask that the labels of the lambdas do not repeat which will come in handy later.

{def:sequentiality_

Definition 6 (Correct term). A multiset of types $[\tau_1, \ldots, \tau_n]$ is **sequential** if the external labels of τ_i and τ_j are different for all $i \neq j$. A typing context Γ is sequential if $\Gamma(x)$ is sequential for every $x \in \operatorname{dom} \Gamma$. The set of **correct** terms is denoted by $\mathcal{T}^{\#}$. A term t is correct if it is typable and it verifies the following three conditions:

1. <u>Uniquely labeled lambdas.</u> If $\lambda^e x.s$ and $\lambda^{e'} y.u$ are subterms of t at different positions, then e and e' must be different labels.

2. <u>Sequential contexts.</u> If s is a subterm of t and $\Gamma \vdash s : \tau$ is derivable, then Γ must be sequential.

3. <u>Sequential types.</u> If s is a subterm of t, the judgment $\Gamma \vdash s : \tau$ is derivable, and there exists a type such that $(\mathcal{M} \xrightarrow{e} \sigma \preceq \Gamma)$ or $(\mathcal{M} \xrightarrow{e} \sigma \preceq \tau)$, then $\mathcal{M}$ must be sequential.

For example, $x^{[\alpha^1] \xrightarrow{2} \beta^3}[x^{\alpha^1}]$ is a correct term. The example we gave above, $(\lambda^1 x.y^{[\alpha^2,\alpha^2] \xrightarrow{3} \alpha^4}[x^{\alpha^2}, x^{\alpha^2}])[a^\alpha$

is not. One problem is that the type of y is not sequential. For a last example, $\lambda^1 x.\lambda^1 y.y^{\alpha^2}$

is not a correct term since labels for lambdas are not unique.

Having defined the types and the terms of our system sheds some light on the resource management capabilities that we claimed it will enjoy: note that we can track very precisely how (i.e. with which type) a term will be used or a bounded variable will be evaluated. This will prove very useful to analyze the λ-calculus.

emma:unique_typing}

Lemma 7 (Unique typing). *Let t be typable, i.e. suppose that there exist a context Γ and a type τ such that $\Gamma \vdash t : \tau$. Furthermore, suppose that t is correct. Then there is a unique derivation that types t. In particular, if $\Gamma' \vdash t : \tau'$, then $\Gamma = \Gamma'$ and $\tau = \tau'$.*

Proof. ♠ By induction on t.   □

2.3   The Calculus

What we want to do know is give the operational semantics for this calculus. The idea is straightforward: in order apply a lambda abstraction to a list of arguments we just replace each occurrence of the variable bounded by the lambda with the corresponding argument. As the term types and is correct we know that we can do it successfully.

Definition 8 (The distributive lambda-calculus). The **distributive lambda-calculus** is given by the set of correct typable terms $\mathcal{T}^{\#}$. For each label $e \in \mathscr{E}$, we define a reduction relation $\xrightarrow{e}_{\#} \subseteq \mathcal{T}^{\#} \times \mathcal{T}^{\#}$ as follows:

$$\mathtt{C}\langle(\lambda^e x.t)\vec{s}\rangle \xrightarrow{e}_{\#} \mathtt{C}\langle t\{x := \vec{s}\}\rangle$$

where $\mathtt{C}$ stands for a **context**. The binary relation $\rightarrow_{\#}$ is the union of all the $\xrightarrow{e}_{\#}$:

$$\rightarrow_{\#} \stackrel{\mathrm{def}}{=} \bigcup_{e \in \mathscr{E}} \xrightarrow{e}_{\#}$$

We sometimes drop the subscript for $\rightarrow_{\#}$, writing just $\rightarrow$, when clear from the context. The set of contexts is given by the grammar:

$$\mathtt{C} ::= \Box \mid \lambda^e x.\mathtt{C} \mid \mathtt{C}\,\vec{t} \mid t[s_1, \ldots, s_{i-1}, \mathtt{C}, s_{i+1}, \ldots, s_n]$$

Contexts can be thought as terms with a single free occurrence of a distinguished variable $\Box$. The notation $\mathtt{C}\langle t \rangle$ stands for the capturing substitution of the occurrence of $\Box$ in $\mathtt{C}$ by t.

In general an n-hole context is a term $\mathtt{C}$ with exactly $n \geq 0$ free occurrences of the distinguished variable $\Box$. If $\mathtt{C}$ is an n-hole context, $\mathtt{C}\langle t_1, \ldots, t_n \rangle$ stands for the term that results from performing the substitution of the i-th occurrence of $\Box_i$ (from left to right) by t_i. If $\mathtt{C}$ is an n-hole contexts for some n, we say that it is a many-hole context.

The following lemmas show that the rewrite rule $\to_\#$ is well-defined.

Lemma 9 (Linearity). *Let $t \in \mathcal{T}^\#$ be a correct term, and let $\Gamma \vdash t : \tau$ be the (unique) type derivation for t. Let x be any variable, and consider the $n \geq 0$ free occurrences of the variable x in the term t, more precisely, write t as $t = \hat{C}\langle x^{\tau_1}, \ldots, x^{\tau_n}\rangle$, where $\hat{C}$ is a context with n-holes such that $x \notin \mathsf{fv}(\hat{C})$. Then $\Gamma(x) = \{\tau_1, \ldots, \tau_n\}$.*

Proof. ♠ By induction on t.   □

Lemma 10 (Subject reduction). *If $\Gamma \vdash C\langle(\lambda^e x.t)\vec{s}\rangle : \tau$ then $\Gamma \vdash C\langle t\{x := \vec{s}\}\rangle : \tau$. Moreover, correctness is preserved.*

Proof. ♠ By induction on C.   □

2.4 Basic Properties

2.5 Termination

2.6 Confluence

2.7 Simulation

2.8 Residual Theory

2.9 Simulation Residuals

Chapter 3

Factorization of Derivations

Conclusions

Appendix

.1 Proof of Lem. 7 — Unique Typing

We will prove that given two typings of a term, they are the same.

Proof. By induction on t.

1. **Variable,** $t = x^\sigma$**.** If $\Gamma \vdash x^\sigma : \tau$, then the last rule in the derivation must be var so it must be the case that $\tau = \sigma$ and $\Gamma = \{x : [\tau]\}$. The same is true for τ' and Γ'. Note that the derivations are equal.

2. **Abstraction,** $t = \lambda y.u$**.** As $\Gamma \vdash \lambda y.u : \tau$, then for some $[\rho_1, \ldots, \rho_n]$ and τ_1 it must be the case that $\tau = [\rho_1, \ldots, \rho_n] \xrightarrow{e} u_1$. Given that the last rule in the derivation must be $\rightarrow_I$, we know that

$$\frac{\Gamma \oplus y : [\rho_1, \ldots, \rho_n] \vdash u : \tau_1}{\Gamma \vdash \lambda^e y.u : [\rho_1, \ldots, \rho_n] \xrightarrow{e} \tau_1} \rightarrow_I$$

The same way, there are $[\rho'_1, \ldots, \rho'_n]$ and τ'_1 such that

$$\Gamma' \oplus y : [\rho'_1, \ldots, \rho'_n] \vdash u : \tau'_1$$

But now we can use our inductive hypothesis, that tells us that $\Gamma \oplus y : [\rho_1, \ldots, \rho_n] = \Gamma' \oplus y : [\rho'_1, \ldots, \rho'_n]$ and $\tau_1 = \tau'_1$, which implies that $\Gamma = \Gamma'$ and $\tau = \tau'$. Moreover, the inductive hypothesis tells us that the derivations are the same.

3. **Application,** $t = u[r_1, \ldots, r_n]$**.** Given that the last rule in the derivation must be $\rightarrow_E$, we know that

$$\frac{\Gamma_0 \vdash u : [\sigma_1, \ldots, \sigma_n] \xrightarrow{e} \tau \quad (\Gamma_i \vdash r_i : \sigma_i)_{i=1}^n}{\Gamma = \Gamma_0 +_{i=1}^n \Gamma_i \vdash u[r_1, \ldots, r_n] : \tau} \rightarrow_E$$

The same way,

$$\frac{\Gamma'_0 \vdash u : [\sigma'_1, \ldots, \sigma'_n] \xrightarrow{e'} \tau' \quad (\Gamma'_i \vdash r_i : \sigma'_i)_{i=1}^n}{\Gamma' = \Gamma'_0 +_{i=1}^n \Gamma'_i \vdash u[r_1, \ldots, r_n] : \tau'} \rightarrow_E$$

15

Using the inductive hypothesis for u, we get that $\Gamma_0 = \Gamma_0'$ and $[\sigma_i]_{i=1}^n \xrightarrow{e} \tau = [\sigma_j']_{j=1}^n \xrightarrow{e'} \tau'$. In particular, $\tau = \tau'$. The inductive hypothesis also tells us that the derivations are the same.

Lastly, using the inductive hypothesis n times for each r_i, we get that $\Gamma_i = \Gamma_i'$ (given that the derivations are the same), so adding everything we can see that $\Gamma_0 +_{i=1}^n \Gamma_i = \Gamma_0' +_{i=1}^n \Gamma_i'$, i.e. $\Gamma = \Gamma'$ (and more generally that the derivations are the same).

$\square$

.2   Proof of Lem. 9 — Linearity

Let $t \in \mathcal{T}^{\#}$ be a correct term, and $\Gamma \vdash t : \tau$ its (unique) type derivation. Let x be any variable, and consider the $n \geq 0$ free occurrences of the variable x in the term t, more precisely, write t as $t = \hat{\mathsf{C}}\langle x^{\tau_1}, \ldots, x^{\tau_n} \rangle$, where $\hat{\mathsf{C}}$ is a context with n-holes such that $x \notin \mathsf{fv}(\hat{\mathsf{C}})$. We will prove that $\Gamma(x) = \{\tau_1, \ldots, \tau_n\}$.

Proof. By induction on t.

1. **Variable (same),** $t = x^\tau$. By uniqueness of derivations, we have that $\{x : \{\tau\}\} \vdash x^\tau : \tau$. Also, $\hat{\mathsf{C}} = \square$ and $t = \hat{\mathsf{C}}\langle x^\sigma \rangle$. And we also have that $\Gamma(x) = \{\tau\}$, so we are done.

2. **Variable (different),** $t = y^\tau$. By uniqueness of derivations, we have that $\{y : \{\tau\}\} \vdash y^\tau : \tau$. Also, $\hat{\mathsf{C}} = y^\tau$ (that is, $\hat{\mathsf{C}}$ does not have any holes) and $t = \hat{\mathsf{C}}$. And we also have that $\Gamma(x) = \varnothing$, so we are done.

3. **Abstraction,** $t = \lambda y.u$. By uniqueness of derivations, we have that the derivation of t is the following.

$$\frac{\Gamma \oplus y : [\rho_1, \ldots, \rho_n] \vdash u : \tau_1}{\Gamma \vdash \lambda^e y.u : [\rho_1, \ldots, \rho_n] \xrightarrow{e} \tau_1} \to_{\mathrm{I}}$$

Given that u is a subterm of t, it is correct. So we may use the inductive hypothesis, which tells us that there is a context $\hat{\mathsf{C}}_1$ such that $u = \hat{\mathsf{C}}_1\langle x^{\tau_1}, \ldots, x^{\tau_n} \rangle$ (where $x \notin \mathsf{fv}(\hat{\mathsf{C}})$) and $(\Gamma \oplus y : [\rho_1, \ldots, \rho_n])(x) = \{\tau_1, \ldots, \tau_n\}$.

Then, we take $\hat{\mathsf{C}}$ to be $\lambda^e y.\hat{\mathsf{C}}_1$. Note that $x \notin \mathsf{fv}(\hat{\mathsf{C}})$, and also that $\Gamma(x) = (\Gamma \oplus y : [\rho_1, \ldots, \rho_n])(x) = \{\tau_1, \ldots, \tau_n\}$.

4. **Application,** $t = u[r_1, \ldots, r_n]$. By uniqueness of derivations, we have that the derivation of t is the following.

$$\frac{\Gamma_0 \vdash u : [\sigma_1, \ldots, \sigma_n] \xrightarrow{e} \tau \qquad (\Gamma_i \vdash r_i : \sigma_i)_{i=1}^n}{\Gamma = \Gamma_0 +_{i=1}^n \Gamma_i \vdash u[r_1, \ldots, r_n] : \tau} \to_{\mathrm{E}}$$

Given that u and all r_i are subterms of t, they are correct; so can use inductive hypothesis on them. By inductive hypothesis on u, we get that $u = \hat{\mathsf{C}}_0\langle x^{\tau_{0,0}}, \ldots, x^{\tau_{0,m_0}}\rangle$ (where $x \notin \mathsf{fv}(\hat{\mathsf{C}}_0)$) and $\Gamma_0(x) = \{\tau_{0,0}, \ldots, \tau_{0,m_0}\}$. On the other hand, by inductive hypothesis on r_i, we get that $r_i = \hat{\mathsf{C}}_i\langle x^{\tau_{i,0}}, \ldots, x^{\tau_{i,m_i}}\rangle$ (where $x \notin \mathsf{fv}(\hat{\mathsf{C}}_i)$) and $\Gamma_i(x) = \{\tau_{i,0}, \ldots, \tau_{i,m_i}\}$.

If we take $\hat{\mathsf{C}}$ to be $\hat{\mathsf{C}}_0[\hat{\mathsf{C}}_1, \ldots, \hat{\mathsf{C}}_n]$, we have that $x \notin \mathsf{fv}(\hat{\mathsf{C}})$ and

$$
\begin{aligned}
t &= u[r_1, \ldots, r_n] \\
&= \hat{\mathsf{C}}_0\langle x^{\tau_{0,0}}, \ldots, x^{\tau_{0,n_0}}\rangle[\hat{\mathsf{C}}_1\langle x^{\tau_{1,0}}, \ldots, x^{\tau_{1,m_1}}\rangle, \ldots \hat{\mathsf{C}}_n\langle x^{\tau_{n,0}}, \ldots, x^{\tau_{n,m_n}}\rangle] \\
&= (\hat{\mathsf{C}}_0[\hat{\mathsf{C}}_1, \ldots, \hat{\mathsf{C}}_n])\langle x^{\tau_{0,0}}, \ldots, x^{\tau_{0,n_0}}, \ldots, x^{\tau_{n,0}}, \ldots, x^{\tau_{n,m_n}}\rangle
\end{aligned}
$$

To end the proof, we check the condition on the context.

$$
\begin{aligned}
\Gamma(x) &= (\Gamma_0 +_{i=1}^n \Gamma_i)(x) \\
&= \Gamma_0(x) +_{i=1}^n \Gamma_i(x) \\
&= \{\tau_{0,0}, \ldots, \tau_{0,m_0}\} +_{i=1}^n \{\tau_{i,0}, \ldots, \tau_{i,m_i}\} \\
&= \{\tau_{0,0}, \ldots, \tau_{0,m_0}, \ldots, \tau_{n,0}, \ldots, \tau_{n,m_n}\}
\end{aligned}
$$

$\square$

.3 Proof of Lem. 10 — Subject Reduction

{subject_reduction_

We are to prove that if $\mathsf{C}\langle(\lambda^e x.t)\vec{s}\rangle$ is correct then $\mathsf{C}\langle t\{x := \vec{s}\}\rangle$ is correct, and, moreover, that their unique typings are under the same typing context and have the same type. Also, we check that correctness is preserved. We need a few auxiliary results:

{lemma:substitution

Lemma 11. *If $\vec{s}$ is a permutation of $\vec{u}$, then $t\{x := \vec{s}\} = t\{x := \vec{u}\}$.*

Proof. By induction on t. $\square$

{lemma:correct_subt

Lemma 12. *If t is correct, then any subterm of t is correct.*

Proof. Note, by definition of correctness (Def. 6), that if t is a correct abstraction $t = \lambda^e x.s$ then s is correct, and if t is a correct application $t = s[u_1, \ldots, u_n]$ then s and $u_1, \ldots, u_n$ are correct. This allows us to conclude by induction on t. $\square$

{lemma:relevance}

Lemma 13 (Relevance). *If $\Gamma \vdash t : \tau$ and $x \in \mathrm{dom}\,\Gamma$ then $x \in \mathsf{fv}(t)$.*

Proof. By induction on t. $\square$

Definition 14. $\Lambda(t)$ stands for the multiset of labels decorating the lambdas of t:

$$
\begin{aligned}
\Lambda(x^\tau) &\overset{\mathrm{def}}{=} [\,] \\
\Lambda(\lambda^e x.t) &\overset{\mathrm{def}}{=} [e] + \Lambda(t) \\
\Lambda(t[s_i]_{i=1}^n) &\overset{\mathrm{def}}{=} \Lambda(t) +_{i=1}^n \Lambda(s_i)
\end{aligned}
$$

For example, $\Lambda((\lambda^1 x.x^{[\alpha^2]\xrightarrow{3}\alpha^2})[\lambda^3 x.x^{\alpha^2}]) = [1,3]$.

Lemma 15. *Let $t, s_1, \ldots, s_n$ be correct terms. Then $\Lambda(t\{x := [s_i]_{i=1}^n\}) = \Lambda(t) +_{i=1}^n \Lambda(s_i)$.*

Proof. By induction on t. □

To prove Lem. 10, we first show that substitution preserves typing, and then that it preserves correctness.

Substitution preserves typing

More precisely, let us show that if $\Gamma \vdash \mathtt{C}\langle(\lambda^e x.t)\vec{s}\rangle : \tau$ is derivable, then $\Gamma \vdash \mathtt{C}\langle t\{x := \vec{s}\}\rangle : \tau$ is derivable. By induction on the context $\mathtt{C}$.

1. **Empty,** $\mathtt{C} = \square$. By induction on t.

   1.1 **Variable (same),** $t = x^\tau$, $\vec{s} = [s]$. We have that $x : [\tau] \vdash x^\tau : \tau$ and $\Delta \vdash s : \sigma$ are derivable, so we are done.

   1.2 **Variable (different),** $t = y^\tau$, $y \neq x$, $\vec{s} = []$. We have that $y : [\tau] \vdash y^\tau : \tau$ is derivable, so we are done.

   1.3 **Abstraction,** $t = \lambda^e y.u$. Let $\Gamma \oplus x : [\sigma_i]_{i=1}^n \vdash \lambda^e x.u : \mathcal{M} \xrightarrow{e} \tau$ be derivable and $\Delta_i \vdash s_i : \sigma_i$ be derivable for all $i = 1..n$. By inversion of the $\to_I$ rule, we have that $\Gamma \oplus y : \mathcal{M} \oplus x : [\sigma_i]_{i=1}^n \vdash u : \tau$ is derivable, so by i.h. $(\Gamma \oplus y : \mathcal{M}) +_{i=1}^n \Delta_i \vdash u\{x := [s_i]_{i=1}^n\} : \tau$ is derivable. Observe that $y \notin \mathsf{fv}(s_i)$ so $y \notin \mathrm{dom}\,\Delta_i$ by Lem. 13. Hence the previous judgment can be written as $(\Gamma +_{i=1}^n \Delta_i) \oplus y : \mathcal{M} \vdash u\{x := [s_i]_{i=1}^n\} : \tau$. Applying the $\to_I$ rule we obtain $\Gamma +_{i=1}^n \Delta_i \vdash \lambda^e y.u\{x := [s_i]_{i=1}^n\} : \mathcal{M} \xrightarrow{e} \tau$ as required.

   1.4 **Application,** $t = u[r_j]_{j=1}^m$. Let $\Gamma \oplus x : [\sigma_i]_{i=1}^n \vdash u[r_j]_{j=1}^m : \tau$ be derivable and $\Delta_i \vdash s_i : \sigma_i$ be derivable for all $i = 1..n$. By inversion of the $\to_E$ rule, the multiset of types $[\sigma_i]_{i=1}^n$ may be partitioned as $[\sigma_i]_{i=1}^n = \sum_{j=0}^m \mathcal{M}_j$, and the typing context Γ may be partitioned as $\Gamma = \sum_{j=0}^m \Gamma_j$ in such a way that $\Gamma_0 \oplus x : \mathcal{M}_0 \vdash u : [\rho_j]_{j=1}^m \xrightarrow{e} \tau$ is derivable and $\Gamma_j \oplus x : \mathcal{M}_j \vdash r_j : \rho_j$ is derivable for all $j = 1..m$. Consider a partition $(\vec{s}_0, \ldots, \vec{s}_j)$ of the list $\vec{s}$ such that for every $j = 0..m$ we have $\ell(\vec{s}_j) = \mathcal{M}_j$. Observe that this partition exists since $\ell(\vec{s}_0 + \ldots + \vec{s}_j) = \ell(\vec{s}) = \sum_{j=0}^m \mathcal{M}_j = [\sigma_i]_{i=1}^n = \ell_x(t)$.

   Moreover, let $\Theta_j = \sum_{i:s_i \in \vec{s}_j} \Delta_i$ for all $j = 0..m$. By i.h. we have that $\Gamma_0 + \Theta_0 \vdash u\{x := \vec{s}_0\} : [\rho_j]_{j=1}^m \xrightarrow{e} \tau$ is derivable and $\Gamma_j + \Theta_j \vdash r_j\{x := \vec{s}_j\} : \rho_j$ is derivable for all $j = 1..m$. Applying the $\to_E$ rule we obtain that $\sum_{j=0}^m \Gamma_j + \sum_{j=0}^m \Theta_j \vdash u[r_j]\{x := \sum_{j=0}^m \vec{s}_j\} : \tau$ is derivable. By definition of $\Gamma_0, \ldots, \Gamma_m$ and $\Theta_0, \ldots, \Theta_m$ this judgment equals $\Gamma +_{i=1}^n \Delta_i \vdash u[r_j]\{x := \sum_{j=0}^m \vec{s}_j\} : \tau$. By definition of $\vec{s}_0, \ldots, \vec{s}_m$ and Lem. 11 this in turn equals $\Gamma +_{i=1}^n \Delta_i \vdash u[r_j]\{x := \vec{s}\} : \tau$, as required.

2. **Under an abstraction,** $C = \lambda^{e'} y . C'$. Straightforward by <u>i.h.</u>.

3. **Left of an application,** $C = C' \vec{u}$. Straightforward by <u>i.h.</u>.

4. **Right of an application,** $C = u[\vec{r}_1, C', \vec{r}_2]$. Straightforward by <u>i.h.</u>.

Substitution preserves correctness

More precisely, let us show that if $C\langle (\lambda^e x.t)\vec{s} \rangle$ is correct then $C\langle t\{x := \vec{s}\} \rangle$ is correct. By induction on C:

1. **Empty,** $C = \square$. Let $\vec{s} = [s_1, \ldots, s_n]$. Observe that if $(\lambda^e x.t)[s_1, \ldots, s_n]$ is correct   `{subject_reduction_` then:

   - [c1] $\Gamma \oplus x : [\sigma_i]_{i=1}^n \vdash t : \tau$ and $\Delta_i \vdash s_i : \sigma_i$ are derivable for all $i = 1..n$,

   - [c2] $t, s_1, \ldots, s_n$ are correct,

   - [c3] there are no free occurrences of x among $s_1, \ldots, s_n$,

   - [c4] all the lambdas occurring in $t, s_1, \ldots, s_n$ have pairwise distinct labels,

   - [c5] $\Gamma +_{i=1}^n \Delta_i$ is a sequential context.

   Condition [c1] holds by inversion of the typing rules, condition [c2] holds by Lem. 12, condition [c3] holds by Barendregt's convention, and conditions [c4] and [c5] hold because the source term is supposed to be correct.

   By induction on t, we check that if $(\lambda^e x.t)[s_1, \ldots, s_n]$ is correct, then $t\{x := \vec{s}\}$ is correct.

   1.1 **Variable (same),** $t = x^\tau$, $\vec{s} = [s]$. Note that $t\{x := \vec{s}\} = s$. Conclude by [c2].

   1.2 **Variable (different),** $t = y^\tau$, $\vec{s} = []$ **with** $x \neq y$. Note that $t\{x := \vec{s}\} = y^\tau$. Conclude by [c2].

   1.3 **Abstraction,** $t = \lambda^{e'} y.u$. Then $\tau = \mathcal{M} \xrightarrow{e'} \rho$ and by inversion $\Gamma \oplus y : \mathcal{M} \vdash$ `{subject_reduction_` $u : \rho$ is derivable. Note that $(\lambda^e x.u)\vec{s}$ is correct, so by <u>i.h.</u> $u\{x := \vec{s}\}$ is correct. The variable y does not occur free in $\vec{s}$, so $(\Gamma \oplus y : \mathcal{M}) +_{i=1}^n \Delta_i = (\Gamma +_{i=1}^n \Delta_i) \oplus (y : \mathcal{M})$. Let us check that $\lambda^{e'} y.u\{x := \vec{s}\}$ is correct:

      1.3.1 <u>Uniquely labeled lambdas.</u> Let e_1 and e_2 be two labels decorating different lambdas of $\lambda^{e'} y.u\{x := \vec{s}\}$, and let us show that $e_1 \neq e_2$. There are two subcases, depending on whether one of the labels decorates the outermost lambda:

      1.3.1.1 **If e_1 or e_2 decorates the outermost lambda.** Suppose without loss of generality that $e_1 = e'$ is the label decorating the outermost lambda. Then by Lem. 15, there are two cases: either e_2 decorates a lambda of u, or e_2 decorates a lambda of some term in the list $\vec{s}$. If e_2 decorates

a lambda of u, then $e_1 \neq e_2$ since we knew that $\lambda^{e'} y.u$ was a correct term by [c2]. If e_2 decorates a lambda of some term in the list $\vec{s}$, then $e_1 \neq e_2$ by condition [c4].

1.3.1.2 **If e_1 and e_2 do not decorate the outermost lambda.** Then e_1 and e_2 decorate different lambdas of the term $u\{x := \vec{s}\}$, and we conclude by <u>i.h.</u>.

1.3.2 <u>Sequential contexts.</u> Let t' be a subterm of $\lambda^{e'} x.u\{x := \vec{s}\}$. If t' is a subterm of $u\{x := \vec{s}\}$, we conclude by <u>i.h.</u>. Otherwise t' is the whole term and the context is $\Gamma +_{i=1}^{n} \Delta_i$, which is sequential by hypothesis [c5].

1.3.3 <u>Sequential types.</u> Let t' be a subterm of $\lambda^{e'} x.u\{x := \vec{s}\}$. If t' is a subterm of $u\{x := \vec{s}\}$, we conclude by <u>i.h.</u>. Otherwise t' is the whole term. Then $\Gamma +_{i=1}^{n} \Delta_i \vdash t' : \tau$ is derivable, since we have already shown that substitution preserves typing. Let $\mathcal{N} \xrightarrow{e''} \phi$ be a type such that $\mathcal{N} \xrightarrow{e''} \phi \preceq \Gamma +_{i=1}^{n} \Delta_i$ or $\mathcal{N} \xrightarrow{e''} \phi \preceq \tau$, and let us show that $\mathcal{N}$ is sequential. In the first case, <u>i.e.</u> if $\mathcal{N} \xrightarrow{e''} \phi \preceq \Gamma +_{i=1}^{n} \Delta_i$ holds, then either $\mathcal{N} \xrightarrow{e''} \phi \preceq \Gamma$ or $\mathcal{N} \xrightarrow{e''} \phi \preceq \Delta_i$ for some $i = 1..n$, and we have that $\mathcal{N}$ is sequential because all the terms $t, s_1, \ldots, s_n$ are correct by [c2]. In the second case, <u>i.e.</u> if $\mathcal{N} \xrightarrow{e''} \phi \preceq \tau$ holds, then we have that $\mathcal{N}$ is sequential because t is correct by [c2].

1.4 **Application,** $t = u\, \vec{r}$. Let $\vec{r} = [r_1, \ldots, r_m]$. Note that $(\lambda^e x.u)\vec{s}_0$ is correct and $(\lambda^e x.r_j)\vec{s}_j$ is correct for all $j = 1..m$, which means that we may apply the <u>i.h.</u> in all these cases. Let us show that $u[r_j]_{j=1}^{m}\{x := \vec{s}\}$ is correct:

1.4.1 <u>Uniquely labeled lambdas.</u> Let e_1 and e_2 be two labels decorating different lambdas of $u[r_j]_{j=1}^{m}\{x := \vec{s}\}$, and let us show that $e_1 \neq e_2$. Observe that the term $u[r_j]_{j=1}^{m}\{x := \vec{s}\} = u\{x := \vec{s}_0\}[r_j\{x := \vec{s}_j\}]_{j=1}^{m}$ has $m + 1$ immediate subterms, namely $u\{x := \vec{s}_0\}$ and $r_j\{x := \vec{s}_j\}$ for each $j = 1..m$. We consider two subcases, depending on whether e_1 and e_2 decorate two lambdas in the same immediate subterm or in different immediate subterms.

1.4.1.1 **The labels e_1 and e_2 decorate the same immediate subterm.** That is, e_1 and e_2 both decorate lambdas in $u\{x := \vec{s}_0\}$ or both decorate lambdas in some $r_j\{x := \vec{s}_j\}$ for some $j = 1..m$. Then we conclude, since both $u\{x := \vec{s}_0\}$ and the $r_j\{x := \vec{s}_j\}$ are correct by <u>i.h.</u>.

1.4.1.2 **The labels e_1 and e_2 decorate different subterms.** Let $r_0 := u$. Then we have that e_1 decorates a lambda in $r_j\{x := \vec{s}_j\}$ for some $j = 0..m$ and e_2 decorates a lambda in $r_k\{x := \vec{s}_k\}$ for some $k = 0..m$, $j \neq k$. By Lem. 15, e_1 decorates a lambda in r_j or a lambda in a term of the list $\vec{s}_j$, and similarly e_2 decorates a lambda in r_k or a lambda in a term

quential_contexts}

_sequential_types}

_case_application}

of the list $\vec{s}_k$. This leaves four possibilities, which are all consequence of [c4].

  1.4.2 <u>Sequential contexts.</u> Similar to item **1.3.2**.

  1.4.3 <u>Sequential types.</u> Similar to item **1.3.3**.

2. **Under an abstraction,** $\text{C} = \lambda^{e'} y.\text{C}'$. Note that $\Gamma \vdash \lambda^{e'} y.\text{C}'\langle t\{x := \vec{s}\}\rangle : \mathcal{M} \xrightarrow{e'} \tau$ is derivable. Let us check the three conditions to see that $\lambda^{e'} y.\text{C}'\langle t\{x := \vec{s}\}\rangle$ is correct:

  2.1 <u>Uniquely labeled lambdas.</u> Any two lambdas in $\text{C}'\langle t\{x := \vec{s}\}\rangle$ have different labels by i.h.. We are left to check that e' does not decorate any lambda in $\text{C}'\langle t\{x := \vec{s}\}\rangle$. Let e_1 be a label that decorates a lambda in $\text{C}'\langle t\{x := \vec{s}\}\rangle$. Then we have that e_1 decorates a lambda in C', or it decorates a lambda in $t\{x := \vec{s}\}$. By what we proved in item **1** this in turn means that it decorates a lambda in t or a lambda in some of the terms of the list $\vec{s}$. In any of these cases we have that $e_1 \neq e'$ since $\lambda^{e'} y.\text{C}'\langle (\lambda^e x.t)\vec{s}\rangle$ is correct.

  2.2 <u>Sequential contexts.</u> Let t' be a subterm of $\lambda^{e'} y.\text{C}'\langle t\{x := \vec{s}\}\rangle$ and let us check that its typing context is sequential. If t' is a subterm of $\text{C}'\langle t\{x := \vec{s}\}\rangle$ we conclude by i.h.. We are left to check the property for t' being the whole term, i.e. that Γ is sequential. By i.h., $\Gamma \oplus y : \mathcal{M}$ is sequential, which implies that Γ is sequential.

  2.3 <u>Sequential types.</u> Let t' be a subterm of $\lambda^{e'} y.\text{C}'\langle t\{x := \vec{s}\}\rangle$ and let us check that, if $\mathcal{N} \xrightarrow{e''} \rho$ is any type occurring in the typing context or in the type of t', then $\mathcal{N}$ is sequential. If t' is a subterm of $\text{C}'\langle t\{x := \vec{s}\}\rangle$ we conclude by i.h.. We are left to check the property for t' being the whole term.

    If $\mathcal{N} \xrightarrow{e''} \rho \preceq \Gamma$, then $\mathcal{N} \xrightarrow{e''} \rho \preceq \Gamma \oplus y : \mathcal{N}$ which is the type of $\text{C}'\langle t\{x := \vec{s}\}\rangle$, so by i.h. $\mathcal{N}$ is sequential.

    If $\mathcal{N} \xrightarrow{e''} \rho \preceq \mathcal{M} \xrightarrow{e''} \tau$, there are three subcases:

  2.3.1 If $\mathcal{N} = \mathcal{M}$, then note that $\mathcal{M}$ is sequential because $\Gamma \oplus y : \mathcal{M}$ is the typing context of $\text{C}'\langle t\{x := \vec{s}\}\rangle$, which is sequential by i.h..

  2.3.2 If $\mathcal{N} \xrightarrow{e''} \rho \preceq \sigma$ where σ is one of the types of $\mathcal{M}$, then $\mathcal{N} \xrightarrow{e''} \rho \preceq \mathcal{M} \xrightarrow{e''} \tau$ which is the typing context of $\text{C}'\langle t\{x := \vec{s}\}\rangle$, and we conclude for this term has sequential types by i.h..

  2.3.3 If $\mathcal{N} \xrightarrow{e''} \rho \preceq \tau$, note that τ is the type of $\text{C}'\langle t\{x := \vec{s}\}\rangle$, and we conclude for this term has sequential types by i.h..

3. **Left of an application,** $\text{C} = \text{C}'\vec{u}$. Note that $\Gamma \vdash \text{C}'\langle t\{x := \vec{s}\}\rangle : [\sigma_j]_{j=1}^{m} \xrightarrow{e'} \tau$ is derivable. Moreover the list of arguments is of the form $\vec{u} = [u_1, \ldots, u_m]$ where all the u_j are correct and $\Delta_j \vdash u_j : \sigma_j$ is derivable for all $j = 1..m$. Then $\Gamma +_{j=1}^{m} \Delta_j \vdash$ {subject_reduction_

$\mathtt{C}'\langle t\{x := \vec{s}\}\rangle[u_j]_{j=1}^m : \tau$ is derivable. Let us check the three conditions to see that $\mathtt{C}'\langle t\{x := \vec{s}\}\rangle[u_j]_{j=1}^m$ is correct:

3.1 <u>Uniquely labeled lambdas.</u> Let e_1 and e_2 be two labels decorating different lambdas in $\mathtt{C}'\langle t\{x := \vec{s}\}\rangle[u_j]_{j=1}^m$. There are three subcases.

   3.1.1 If e_1 and e_2 both decorate lambdas in the subterm $\mathtt{C}'\langle t\{x := \vec{s}\}\rangle$ then $e_1 \neq e_2$ since $\mathtt{C}'\langle t\{x := \vec{s}\}\rangle$ is correct by <u>i.h.</u>.

   3.1.2 If e_1 and e_2 both decorate lambdas somewhere in $[u_1, \ldots, u_m]$ then $e_1 \neq e_2$ since $\mathtt{C}'\langle t\rangle[u_1, \ldots, u_m]$ is correct by hypothesis.

   3.1.3 If e_1 decorates a lambda in $\mathtt{C}'\langle t\{x := \vec{s}\}\rangle$ and e_2 decorates a lambda in one of the terms u_j for some $j = 1..m$, then note that e_1 must either decorate a lambda in $\mathtt{C}'$ or a lambda in $t\{x := \vec{s}\}$. By what we proved in item **1** this in turn means that it decorates a lambda in t or a lambda in some of the terms of the list $\vec{s}$. In any of these cases we have that $e_1 \neq e_2$ since $\mathtt{C}'\langle (\lambda^e x.t)\vec{s}\rangle[u_j]_{j=1}^m$ is correct by hypothesis.

3.2 <u>Sequential contexts.</u> Let t' be a subterm of $\mathtt{C}'\langle t\{x := \vec{s}\}\rangle[u_j]_{j=1}^m$ and let us show that its typing context is sequential. If t' is a subterm of $\mathtt{C}'\langle t\{x := \vec{s}\}\rangle$ we conclude by <u>i.h.</u>. If t' is a subterm of one of the u_j for some $j = 1..m$, we conclude using that u_j is correct by hypothesis. It remains to check that the whole term is correct, <u>i.e.</u> that $\Gamma +_{j=1}^m \Delta_j$ is sequential. Observe that $\Gamma +_{j=1}^m \Delta_j$ is also the typing context of $\mathtt{C}'\langle (\lambda^e x.t)\vec{s}\rangle[u_j]_{j=1}^m$, which is correct by hypothesis.

3.3 <u>Sequential types.</u> Let t' be a subterm of $\mathtt{C}'\langle t\{x := \vec{s}\}\rangle[u_j]_{j=1}^m$ and let us show if $\mathcal{N} \xrightarrow{e'} \rho$ is a type that occurs in the typing context or in the type of t', then $\mathcal{N}$ is sequential. If t' is a subterm of $\mathtt{C}'\langle t\{x := \vec{s}\}\rangle$ we conclude by <u>i.h.</u>. If t' is a subterm of one of the u_j for some $j = 1..m$, we conclude using that u_j is correct by hypothesis. We are left to check the property for t' being the whole term.

If $\mathcal{N} \xrightarrow{e'} \rho \preceq \Gamma +_{j=1}^m \Delta_j$, we conclude by observing that $\Gamma +_{j=1}^m \Delta_j$ is also the typing context of $\mathtt{C}'\langle (\lambda^e x.t)\vec{s}\rangle[u_j]_{j=1}^m$, which is correct by hypothesis, so it has sequential types.

Similarly, if $\mathcal{N} \xrightarrow{e'} \rho \preceq \tau$, we conclude by observing that τ is also the type of $\mathtt{C}'\langle (\lambda^e x.t)\vec{s}\rangle[u_j]_{j=1}^m$.

4. **Right of an application,** $\mathtt{C} = u[\vec{r}_1, \mathtt{C}', \vec{r}_2]$**.** Similar to item **3**.

Bibliography

[ABKL14]   Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In POPL '14, San Diego, CA, USA, January 20-21, 2014, pages 659–670, 2014. 2

[AL13]   Andrea Asperti and Jean-Jacques Lévy. The cost of usage in the lambda-calculus. In 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013, pages 293–300, 2013. iv

[Bar84]   Henk Barendregt. The Lambda Calculus: Its Syntax and Semantics, volume 103. Elsevier, 1984. iv

[BKDR14]   Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. The inhabitation problem for non-idempotent intersection types. In IFIP International Conference on Theoretical Computer Science, pages 341–354. Springer, 2014. v, vi, 5

[BKV17]   Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. Logic Journal of the IGPL, 25(4):431–464, 2017. v, vi, vii, 5

[BL13]   Alexis Bernadet and Stéphane Jean Lengrand. Non-idempotent intersection types and strong normalisation. arXiv preprint arXiv:1310.1622, 2013. v

[Bou93]   Gérard Boudol. The lambda-calculus with multiplicities. In CONCUR'93, pages 1–6. Springer, 1993. v

[Car07]   Daniel de Carvalho. Sémantiques de la logique linéaire et temps de calcul. PhD thesis, Ecole Doctorale Physique et Sciences de la Matière (Marseille), 2007. vi

[CD78]   Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for lambda-terms. Arch. Math. Log., 19(1):139–156, 1978. v

[CF58]   H.B. Curry and R. Feys. <u>Combinatory Logic</u>. Number v. 1 in Combinatory
           Logic. North-Holland Publishing Company, 1958. iii

[CR36]   Alonzo Church and J Barkley Rosser.   Some properties of conversion.
           <u>Transactions of the American Mathematical Society</u>, 39(3):472–482, 1936. iii

[DJK91]   Nachum Dershowitz, Jean-Pierre Jouannaud, and Jan Willem Klop. Open
           problems in rewriting. In <u>International Conference on Rewriting Techniques</u>
           <u>and Applications</u>, pages 445–456. Springer, 1991. iv

[Ehr12]   Thomas Ehrhard.   Collapsing non-idempotent intersection types.   In
           <u>LIPIcs-Leibniz International Proceedings in Informatics</u>, volume 16. Schloss
           Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012. v

[ER03]   Thomas Ehrhard and Laurent Regnier.   The differential lambda-calculus.
           <u>Theoretical Computer Science</u>, 309(1):1–41, 2003. v

[Gar94]   Philippa Gardner.   Discovering needed reductions using type theory.   In
           <u>Theoretical Aspects of Computer Software</u>, pages 555–574. Springer, 1994.
           vi

[Gir87]   Jean-Yves Girard. Linear logic. <u>Theoretical computer science</u>, 50(1):1–101,
           1987. v

[Kes16]   Delia Kesner.   Reasoning about call-by-need by means of types.
           In <u>International Conference on Foundations of Software Science and</u>
           <u>Computation Structures</u>, pages 424–441. Springer, 2016. v

[KL07]   Delia Kesner and Stéphane Lengrand.   Resource operators for λ-calculus.
           <u>Information and Computation</u>, 205(4):419–473, 2007. v

[KR]   Delia Kesner and Fabien Renaud. The prismoid of resources. Springer. v

[KRV18]   Delia Kesner, Alejandro Ríos, and Andrés Viso. Call-by-need, neededness
           and all that. In <u>21st International Conference on Foundations of Software</u>
           <u>Science and Computation Structures (FoSSaCS)</u>, 2018. v

[Lan94]   Cosimo Laneve.   Distributive evaluations of λ-calculus.   <u>Fundamenta</u>
           <u>Informaticae</u>, 20(4):333–352, 1994. iv

[Lév78]   Jean-Jacques Lévy. <u>Réductions correctes et optimales dans le lambda-calcul</u>.
           PhD thesis, Université de Paris 7, 1978. iii, iv

[Lev15]   Jean-Jacques Levy. Redexes are stable in the λ-calculus. 27:1–13, 07 2015. iv

[Mel96]   Paul-André Melliès. <u>Description Abstraite des Systèmes de Réécriture</u>. PhD
           thesis, Université Paris 7, december 1996. iv, 2, 3

[Mel97]   Paul-André Melliès.   A factorisation theorem in rewriting theory.   In Category Theory and Computer Science, 7th International Conference, CTCS '97, Santa Margherita Ligure, Italy, September 4-6, 1997, Proceedings, pages 49–68, 1997. iii, v, vii

[Mel00]   Paul-André Melliès. Axiomatic rewriting theory II: the $\lambda\sigma$-calculus enjoys finite normalisation cones. J. Log. Comput., 10(3):461–487, 2000. iii

[Mel02a]   Paul-André Melliès. Axiomatic rewriting theory VI residual theory revisited. In Sophie Tison, editor, Rewriting Techniques and Applications, 13th International Conference, RTA 2002, Copenhagen, Denmark, July 22-24, 2002, Proceedings, volume 2378 of Lecture Notes in Computer Science, pages 24–50. Springer, 2002. iii

[Mel02b]   Paul-André Mellies. Axiomatic rewriting theory vi: Residual theory revisited. In Rewriting techniques and applications, pages 5–11. Springer, 2002. iii

[Mel05]   Paul-André Melliès. Axiomatic rewriting theory I: A diagrammatic standardization theorem. In Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday, pages 554–638, 2005. iii

[Ter03]   Terese.   Term Rewriting Systems, volume 55 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003. iv, vii

[Via17]   Pierre Vial.   Non-Idempotent Typing Operators, Beyond the Lambda-Calculus. PhD thesis, Université Paris 7, december 2017. v

[Zil84]   Marisa Venturini Zilli. Reduction graphs in the lambda calculus. Theor. Comput. Sci., 29:251–275, 1984. iv