UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Matemática

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 $(A, B) \to \ldots \to (A', B')$. These sequences can always be decomposed as two non-interfering computations $A \to \ldots \to A'$ and $B \to \ldots \to B'$. 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 $RS'T'$:

$$\begin{tikzcd}
(\lambda x.xx)(Kz) \arrow[rr, "R"] \arrow[ddr, "S"'] & & (\lambda x.xx)(\lambda y.z) \arrow[dr, "S'"] & & \\
& & (Kz)(\lambda y.z) \arrow[r, "R'_2"] & (\lambda y.z)(\lambda y.z) \arrow[dr, "T'"] & \\
& (Kz)(Kz) \arrow[ur, "R_1"] \arrow[r, "R_2"] & (\lambda y.z)(Kz) \arrow[ur, "R'_1"] \arrow[rr, "T"] & & z
\end{tikzcd}$$

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:

$$\begin{tikzcd}
 & (\lambda x.a)((\lambda x.b)\Omega) \arrow[d, "T_1"'] \arrow[dddl, bend right, "R"'] \arrow[dddr, bend left, "S"] & \\
 & (\lambda x.a)((\lambda x.b)\Omega) \arrow[d, "T_2"'] \arrow[ddl, bend right] \arrow[ddr, bend left] & \\
 & (\lambda x.a)((\lambda x.b)\Omega) \arrow[d, "T_3"'] \arrow[dl, bend right] \arrow[dr, bend left] & \\
a & \vdots & (\lambda x.a)b
\end{tikzcd}$$

Observe that for all $n \in \mathbb{N}$ we have that $T_1 \dots T_n \sqsubseteq R$ and $T_1 \dots T_n \sqsubseteq S$ 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 external and one of internal derivations.

The difficulty to understand derivation spaces is due to three pervasive phenomena of interaction between computations. The first phenomenon is duplication: 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 residuals of R, and, conversely, R is an ancestor of R_1 and R_2. The second phenomenon is erasure: in the diagram above, the step T erases the step R_1', resulting in no copies of R_1'. The third phenomenon is creation: 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 resources. 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 non-idempotent intersection type systems, 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 normalization, 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 intersection type constructor $\tau \cap \sigma$. Non-idempotent 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 $\mathcal{W}$)**.**

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

Observe that the $\to_{\mathrm{E}}$ 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 $\to_{\mathrm{I}}$ rule, and $t[s_1, \ldots, s_n]$ for an application of the $\to_{\mathrm{E}}$ rule, so that, for example, $\lambda x.x^{[\alpha,\alpha]\to\beta}[x^{\alpha}, x^{\alpha}]$ represents the following typing derivation:

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

The second obstacle is a major one for our purposes: proof normalization 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:

$$\begin{tikzcd}
(\lambda x.y^{[\alpha]\to[\alpha]\to\beta}[x^{\alpha}][x^{\alpha}])[z^{[\gamma]\to\alpha}[z^{\gamma}], z^{[]\to\alpha}[]] \arrow[r] \arrow[dr] & y^{[\alpha]\to[\alpha]\to\beta}[z^{[\gamma]\to\alpha}[z^{\gamma}]][z^{[]\to\alpha}[]] \\
 & y^{[\alpha]\to[\alpha]\to\beta}[z^{[]\to\alpha}[]][z^{[\gamma]\to\alpha}[z^{\gamma}]]
\end{tikzcd}$$

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 2.6, 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 garbage if it maps to an empty derivation in the $\lambda^{\#}$-calculus. This gives rise to an orthogonal notion of garbage-free derivation. The notion of garbage-free derivation is closely related with the notions of needed step [Ter03, Section 8.6], typed occurrence of a redex [BKV17], and external derivation [Mel97]. Using this notion of garbage we prove a factorization theorem 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

{ch: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** $\bot \in A$, 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. $f(\bot) = \bot$, 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 $\mathsf{src}, \mathsf{tgt} : \mathsf{Stp} \to \mathsf{Obj}$ indicating the source and target of each step, and a **residual function** $(/)$ such that given any two steps $R, S \in \mathsf{Stp}$ with the same source, yields a set of steps R/S such that $\mathsf{src}(R') = \mathsf{tgt}(S)$ 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 $\mathsf{tgt}(\rho) = \mathsf{src}(\sigma)$ 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 $R \in \mathsf{Stp}$.

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

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

4. Semantic Orthogonality. Let $R, S \in \mathsf{Stp}$ 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 $T \in \mathsf{Stp}$ 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

{ch:lambdadist}

In this chapter we present a distributive λ-calculus ($\lambda^{\#}$), 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 labels 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].

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:

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

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

In a type like α^e and $\mathcal{M} \overset{e}{\to} \tau$, 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:

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

The notation $\Gamma \oplus \Delta$ stands for the **disjoint sum of contexts**, i.e. it stands for $\Gamma + \Delta$ provided $\operatorname{dom}\Gamma \cap \operatorname{dom}\Delta = \varnothing$. 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 $\operatorname{dom}(x : \mathcal{M}) = \{x\}$.

Remark that the (only) difference with the system $\mathcal{W}$ is that we include labels. The other, related, difference will appear when we define the terms of the calculus.

More in general, 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.

2.2 Syntax

Definition 4 (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.

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

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

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 problem: 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 each 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.

That invariant is called correctness, and we will consider $\lambda^\#$ to be the system of all correct terms.

Definition 5 (Correct term)**.** A typable term t is **correct** if the three following conditions hold:

- **Unique lambdas.** Lambdas in a term are decorated with pairwise distinct labels.
- **Sequential contexts.** For every judgment $\Gamma \vdash s : \tau$ in the type derivation of t, and for every variable x, the set $\Gamma(x)$ is sequential.
- **Sequential types.** For every subtype $\mathcal{M} \overset{e}{\to} \tau$ ocurring somewhere in the derivation of t, the set $\mathcal{M}$ is sequential.

The set of correct terms is written $\lambda^{\#}$.

Having defined the types and the terms of out system sheds some light on the resource management capabilities that we claimed it will enjoy: note that we can track very precisely how a bounded variable will be used.

2.3 Basic Properties

2.4 Termination

2.5 Confluence

2.6 Simulation

{section:simulatio

2.7 Residual Theory

2.8 Simulation Residuals

Chapter 3

Factorization of Derivations

{ch:factorization}

3.1 Garbage

3.2 Sieving

3.3 Some Properties

3.4 Factorization of Garbage

3.5 Lattices

Conclusions

{ch:conclusions}

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. Combinatory Logic. Number v. 1 in Combinatory Logic. North-Holland Publishing Company, 1958. iii

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

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

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

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

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

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

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

[KL07] Delia Kesner and Stéphane Lengrand. Resource operators for λ-calculus. Information and Computation, 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 21st International Conference on Foundations of Software Science and Computation Structures (FoSSaCS), 2018. v

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

[Lév78] Jean-Jacques Lévy. Réductions correctes et optimales dans le lambda-calcul. 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. Description Abstraite des Systèmes de Réécriture. 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