Intersection Types for the λ-Calculus

Federico Olimpieri

1 Introduction

The writing of these notes has been inspired by a series of seminars that I gave for the Category theory working group (*Category Theory Lunch*) at the School of Mathematics of the University of Leeds.

2 Preliminaries

2.1 Integers

We consider the category $\mathbb{O}_f$ where object are *finite ordinals* $[n] = \{1, \ldots, n\}$, for $n \in \mathbb{N}$, and morphisms are functions. The category $\mathbb{O}_f$ is symmetric strict monoidal (cocartesian in particular), with tensor product given by addition: $[n] \oplus [m] = [n+m]$. We set

$$\mathbb{O}^l_f([m],[n]) = \{\alpha \in \mathbb{O}_f([m],[n]) \mid \alpha \text{ is bijective.}\} \qquad \mathbb{O}^a_f([m],[n]) = \{\alpha \in \mathbb{O}_f([m],[n]) \mid \alpha \text{ is injective.}\}$$

$$\mathbb{O}^r_f([m],[n]) = \{\alpha \in \mathbb{O}_f([m],[n]) \mid \alpha \text{ is surjective.}\} \qquad \mathbb{O}^c_f([m],[n]) = \mathbb{O}_f([m],[n])$$

Then evidently the parametric family of sets $\mathbb{O}^\spadesuit_f([m],[n])$ for $m, n \in \mathbb{N}$, and $\spadesuit \in \{l, a, r, c\}$ determine full subcategories of $\mathbb{O}_f$, that we denote as $\mathbb{O}^\spadesuit_f$.

2.2 Lists, Multisets ans Sets

Let $\mathcal{X}$ be a preordered set. From $\mathbb{O}^\spadesuit_f$ we can build preorders of indexed families of objects over finite ordinals. Let $\langle a_1, \ldots, a_k\rangle$ be a list of elements of $\mathcal{X}$. We write $\mathsf{len}(\vec{a})$ for its length. We denote lists as $\vec{a}, \vec{b}, \vec{c}\ldots$ Given a list $\vec{a} = \langle a_1, \ldots, a_k\rangle$ and a function $\alpha : [k] \to [k']$ we define the *right action of α on $\vec{a}$* as $\vec{a}\{\alpha\} = \langle a_{\alpha(1)}, \ldots, a_{\alpha(k)}\rangle$. We define the category $\mathcal{X}^\spadesuit$ of $\spadesuit$-*lists* over $\mathcal{X}$, as follows:

1. $|\mathbb{O}^\spadesuit_f\mathcal{X}| = \{\langle a_1, \ldots, a_n\rangle \mid a_i \in \mathcal{X}\}$.
2. The preorder relation is the smallest one generated by the following rule:

$$\frac{\alpha \in \mathbb{O}^\spadesuit_f([m],[n]) \qquad a_{\alpha(i)} \leq_{\mathcal{X}} b_i}{\langle a_1, \ldots, a_n\rangle \leq_{\mathbb{O}^\spadesuit_f\mathcal{X}} \langle b_1, \ldots, b_m\rangle}$$

The preorder $\mathbb{O}_f\mathcal{X}$ is equipped with a monoidal structure given by lists concatenation $\vec{a} :: \vec{b}$.

Let X be a set. We denote as $\mathcal{M}_f(X)$ the *free commutative monoid* over X. Elements of $\mathcal{M}_f(X)$ are *multisets of elements of* X, that we denote as $\overline{a} = [a_1, \ldots, a_k] \in \mathcal{M}_f$, with $a_i \in X$. We denote as $\wp_f(X)$ the set of *finite subsets* of X. We denote finite subsets as $\tilde{a} = \{a_1, \ldots, a_k\}$ with $a_i \in X$.

3 Pure λ-Calculus

For a proper introduction to the theory of pure λ-calculus we refer to [4, 7]. We fix a countable set of variables $\mathcal{V} \ni x, y, z \dots$ The λ-*terms* are inductively defined *via* the following grammar:

$$\Lambda \ni M, N ::= x \mid \lambda x.M \mid MN$$

As usual, application associates to the left, and has higher precedence than abstraction. E.g., $\lambda x.\lambda y.\lambda z.xyz := \lambda x.(\lambda y.(\lambda z.((xy)z)))$. We let $M\vec{N}$ (resp. $\lambda\vec{x}.M$) denote $MN_1 \cdots N_k$ (resp. $\lambda x_1. \dots \lambda x_n.M$). The *free variables* of a term M are defined by induction on the structure of M as follows:

$$\mathrm{FV}(x) = \{x\} \qquad \mathrm{FV}(\lambda x.M) = \mathrm{FV}(M) \setminus \{x\} \qquad \mathrm{FV}(PQ) = \mathrm{FV}(P) \cup \mathrm{FV}(Q)$$

If $\mathrm{FV}(M) = \emptyset$ then M is *closed.* We consider λ-terms up to renaming of bound variables. This can be made formal by the explicit definition of α-equivalence (see [7]) or by using De Brujin indexes/levels[1]. If $\mathrm{FV}(M) = \{x_1, \dots, x_n\}$ then the *closure* of M is the term $\lambda x._1 \dots \lambda x._n M$.

Example 3.1. *Some famous terms:*

$$I := \lambda x.x \qquad \Delta := \lambda x.xx \qquad \Omega := \Delta\Delta \qquad R := \lambda x.f(xx) \qquad Y := \lambda f.RR$$

$$\underline{n} = \lambda f.\lambda x.f^n x \qquad \textit{for } n \in \mathbb{N}$$

$$R' = \lambda x.\lambda y.y(xxy) \qquad T = R'R'$$

I is called the identity*; Y is called* Curry's fixedpoint operator*; T is called* Turing's fixedpoint operator*; $\underline{n}$ is the* Church numeral *associated to the integer $n \in \mathbb{N}$.*

3.1 Basic Notions of Substitution

Given $M, N \in \Lambda, x \in \mathcal{V}$ we define $M\{N/x\} \in \Lambda$, the *substitution of x by N in M*, by induction on the structure of M as follows:

$$M\{N/x\} = \begin{cases} N & \text{if } y = x \\ y & \text{otherwise.} \end{cases} \qquad \lambda y.M\{N/x\} = \begin{cases} \lambda y.M & \text{if } x = y \\ \lambda y.(M\{N/x\}) & \text{otherwise.} \end{cases}$$

$$PQ\{N/x\} = P\{N/x\}Q\{N/x\}$$

Given $M, N_1, \dots, N_n \in \Lambda, x_1, \dots, x_n \in \mathcal{V}, x_i \neq x_j$ we define also the *parallel substitution*

$$M\{N_1, \dots, N_n/x_1, \dots, x_n\},$$

by induction on the structure of M as follows:

$$x\{N_1, \dots, N_n/x_1, \dots, x_n\} = \begin{cases} N_i & \text{if } x = x_i \text{ for some } i \in [n] \\ x & \text{otherwise.} \end{cases}$$

$$\lambda x.M\{N_1, \dots, N_n/x_1, \dots, x_n\} = \begin{cases} \lambda x.M & \text{if } x = x_i \text{ for some } i \in [n] \\ \lambda x.(M\{N_1, \dots, N_n/x_1, \dots, x_n\}) & \text{otherwise.} \end{cases}$$

$$PQ\{N_1, \dots, N_n/x_1, \dots, x_n\} = P\{N_1, \dots, N_n/x_1, \dots, x_n\}Q\{N_1, \dots, N_n/x_1, \dots, x_n\}$$

[1] These last two choices amount to taking representative of the α-equivalence classes.

3.2 β-Reduction

We define the *one-step β-reduction relation* $\to_\beta \subseteq \Lambda^2$ by induction as follows.

Root-step:

$$\frac{}{(\lambda x.M)N \to_\beta M\{N/x\}}$$

Contextual extension:

$$\frac{M \to_\beta M'}{\lambda x.M \to_\beta \lambda x.M'} \qquad \frac{M \to_\beta M'}{MN \to_\beta M'N} \qquad \frac{N \to_\beta N'}{MN \to_\beta MN'}$$

We denote as $\to^* \subseteq \Lambda^2$ the transitive and reflexive closure of $\to_\beta$. We denote as $=_\beta \subseteq \Lambda^2$ the smallest equivalence relation generated by $\to^*$. The β-reduction is the computation rule of λ-calculus. It should be thought of as an *execution procedure* of functional programs.

Example 3.2. *We give some examples of reductions.*

$$(\lambda x.x)M \to_\beta M \qquad \Delta M \to_\beta MM \qquad \Omega \to_\beta \Omega$$
$$Y \to_\beta \lambda f.fRR \qquad YM \to_\beta M(\lambda x.M(xx))(\lambda x.M(xx)) =_\beta M(YM)$$

$$T \to_\beta \lambda y.y(Ty) \qquad TM \to^* M(TM)$$

It is worth noting that while Curry's operator Y validates the fixedpoint rule

$$YM = MYM$$

by the means of β-equality, Turing's one validates it just by β-reduction

$$TM \to^* MTM.$$

3.3 Normalization Properties

A program produces an *output* when its execution on a given *input* terminates. This is modeled in the λ-calculus *via* the notion of *normalization.* A term of the shape $(\lambda x.M)N$ is called a *redex.* This kind of terms should be thought of as functions applied to an argument, attending evaluation. A *normal form* is a λ-term that does not contain any redex as subterm. We denote as $\mathsf{NF}(\Lambda) \subseteq \Lambda$ the set of normal forms. A λ-term M is *normalizable* when there exists $N \in \mathsf{NF}(\Lambda)$ such that $M \to^* N$. A term M is *strongly normalizable* if there is no infinite sequence of reduction steps starting from M.

Proposition 3.3. *Let $M \in \Lambda$. There exist $x_1, \ldots, x_n \in \mathcal{V}, Q, Q_1, \ldots, Q_m \in \Lambda$ such that*

$$M = \lambda x_1. \ldots \lambda x_n.QQ_1 \ldots Q_m$$

with Q being either a variable or an abstraction.

We have the following inductive characterization of normal forms:

$$\mathsf{NF}(\Lambda) \ni P ::= \lambda x_1. \ldots \lambda x_n.Q \qquad Q ::= xP_1 \ldots P_m$$

Proposition 3.3 gives a handy characterization of λ-terms, depending on the shape of their 'heads'. If the head of a term is a variable, this intuitively means that some partial output has been computed. This intuition leads to the definition of *head-normal forms*:

$$\mathsf{HNF}(\Lambda) \ni P ::= \lambda x_1. \ldots \lambda x_n.Q \qquad Q ::= xM_1 \ldots M_m$$

where $M_i \in \Lambda$. Trivially, any normal form is a head-normal form, the converse does not hold in general. We say that a term M is *head-normalizable* if there exists $N \in \mathsf{HNF}(\Lambda)$ such that $M \to^* N$.

Example 3.4. *Some examples of normalization.*

1. *Ω is neither normalizable nor head-normalizable.*
2. *The term $(\lambda z.x)\Omega$ is normalizable, with normal form x. This term is clearly not strongly normalizable, since we have the infinite reduction chain consisting of keep firing the redex of Ω.*
3. *T and Y are both head-normalizable, but they are not normalizable.*

$$T \to \lambda y.y(Ty) \in \mathsf{HNF}(\Lambda) \to_\beta \ldots \;\; \lambda y.y(y(\ldots(Ty)\ldots)) \in \mathsf{HNF}(\Lambda) \to \ldots$$

$$Y \to \lambda f.fRR \in \mathsf{HNF}(\Lambda) \to \lambda f.f(f(\ldots(RR)\ldots)) \in \mathsf{HNF}(\Lambda) \to \ldots.$$

3.3.1 Reduction Strategies

As should be clear by now, the β-reduction models evaluation but does not say anything about the *order* of computation steps. However, the order in which the evaluation is carried out is crucial, as shown by the simple example of $(\lambda x.z)\Omega$, where choosing the *leftmost* redex as the first one to compute gives the normal form of the term.

We shall now define a *reduction strategy* for λ-terms, called the *head-reduction*. Intuitively, this strategy consists of firing the redexes that appear at 'head position' in the body of the considered term. In order to define it, we will exploit the characterization of terms given by Proposition 3.3.

Definition 3.5 (Head-Reduction Strategy)**.** *We define a function $H : \Lambda \to \Lambda$, called the* head-reduction strategy*, by cases as follows:*

$$H(M) = \begin{cases} \lambda x_1. \ldots \lambda x_m.P\{Q/x\}Q_1 \ldots Q_n & \textit{if } M = \lambda x_1. \ldots \lambda x_m.(\lambda x.P)QQ_1 \ldots Q_n \\ M & \textit{otherwise}\ . \end{cases}$$

We remark that $M \to^* H(M)$. We set $H^n(M)$ to denote the n-th iteration of H on M, assuming that $H^0(M) = M$. We say that the *head-reduction for M ends* if there exists $n \in \mathbb{N}$ s.t. $H^n(M) \in \mathsf{HNF}(\Lambda)$. Trivially, if the head-reduction for M ends then M is head-normalizable.

3.4 Confluence of β

An important property of the β-reduction is confluence, which assures uniqueness of normal forms.

Theorem 3.6 (Church-Rosser/Confluence)**.** *Let $M \to^* N_1$ and $M \to^* N_2$. Then there exists $N \in \Lambda$ such that $N_1 \to^* N$ and $N_2 \to^* N$.*

A proof of theorem by Tait and Martin Löf can be found in Chapter 3.2 of [4]. The main ingredient of the proof is the definition of an auxiliary reduction, called the *parallel reduction* (where one can fire more than one redex at once).

4 Solvability and Böhm Trees

The λ-terms are classified into solvable/unsolvable, depending on their capability of interaction with the environment.

Definition 4.1. *A closed λ-term N is* solvable *if there are $\vec{P} \in \Lambda$ such that $N\vec{P} \to^* \mathsf{I}$. A λ-term M is* solvable *if its closure $\lambda\vec{x}.M$ is solvable. Otherwise M is called* unsolvable.

Theorem 4.2 ([9])**.** *A λ-term M is solvable if and only if M has an hnf.*

The typical examples of unsolvables are Ω and YI. The execution of a λ-term can be represented as a possibly infinite tree, obtained by collecting all the stable pieces of information coming out from the computation (if any). The complete lack of information is represented by a constant $\bot$.

Definition 4.3. *The* Böhm tree $\mathrm{BT}(M)$ *of a λ-term M is (possibly infinite) labelled tree defined coinductively as follows:*

- *if $M \to^* \lambda x_1 \dots x_m.xQ_1 \cdots Q_n$ (for $n, m \geq 0$), then*

$$\mathrm{BT}(M) = \begin{array}{c} \lambda x_1 \dots x_m.x_i \\ \mathrm{BT}(Q_1) \quad \cdots \quad \mathrm{BT}(Q_n), \end{array}$$

- *otherwise M is unsolvable and* $\mathrm{BT}(M) = \bot$.

Example 4.4. *The following are examples of Böhm trees.*

1. $\mathrm{BT}(\mathsf{I}) = \lambda x.x$, $\mathrm{BT}(1) = \lambda xy.xy$ *and* $\mathrm{BT}(\Delta) = \lambda x.xx$.
2. *More generally, if M is in β-nf then* $\mathrm{BT}(M) = M$.
3. *Since Ω is unsolvable, we have* $\mathrm{BT}(\Omega) = \bot$.
4. *More interestingly,* $\mathrm{BT}(\mathsf{Y}) = \lambda f.f(f(f(f(f(\cdots)))))$.

Remark 4.5. *Since* $\mathrm{BT}(M)$ *is defined coinductively, so is the equality between Böhm trees. That is,* $\mathrm{BT}(M_1) = \mathrm{BT}(M_2)$ *if and only if either M_1, M_2 are both unsolvable, or (for $i = 1, 2$) $M_i \twoheadrightarrow_h \lambda\vec{x}.yN_{i1} \cdots N_{ik}$ where* $\mathrm{BT}(N_{1j}) = \mathrm{BT}(N_{2j})$*, for all j.*

The equivalence $\mathcal{B}$ obtained by equating all λ-terms having the same Böhm tree, i.e.

$$\mathcal{B} = \{(M, N) \mid \mathrm{BT}(M) = \mathrm{BT}(N)\} \subseteq \Lambda^2,$$

is an example of a so-called λ-theory, namely an equational theory of λ-calculus. These theories become the main object of study when considering the computational equivalence more important than the process of computation itself [8].

4.1 A Theory of Program Approximation

Another approach to the notion of Böhm tree is given by the *theory of finite approximants.* Intuitively, the finite approximants of a λ-term M are obtained by cutting its Böhm tree into finite pieces, replacing the removed subtree with $\bot$. A finite approximant is formally defined as a term in normal form living in a λ-calculus extended with a constant $\bot$.

Definition 4.6. 1. *The set* $\Lambda_\perp$ *of* $\lambda_\perp$-terms *is inductively defined by the simplified grammar:*

$$\Lambda_\perp \ni M, N, L ::= \ \perp \ \mid x \ \mid \ \lambda x.M \ \mid \ MN$$

2. *Let* $\leq_\perp \subseteq \Lambda_\perp \times \Lambda_\perp$ *denotes the least contextual closed preorder generated by setting*

$$\perp \leq M, \text{ for all } M \in \Lambda_\perp.$$

3. *The* $\lambda_\perp$*-terms are endowed with the reduction* $\to_{\beta\perp}$*, namely* β*-reduction extended with*

$$\begin{array}{rcll} \lambda x.\perp & \to_\perp & \perp, & \\ \perp M_1 \cdots M_n & \to_\perp & \perp & (\text{for } n > 0). \end{array}$$

4. *The set* $\mathcal{A} \subseteq \Lambda_\perp$ *of* finite approximants *is defined by:*

$$\mathcal{A} \ni P, Q ::= \ \perp \ \mid \ \lambda x_1 \dots x_n.yP_1 \cdots P_k \quad (\text{for } n, k \geq 0)$$

5. *Given a* λ*-term* M*, the set* $\mathcal{A}(M)$ *of* finite approximants of M *is defined as follows:*

$$\mathcal{A}(M) = \{P \in \mathcal{A} \mid \exists N \in \Lambda \,.\, M \to^* N \text{ and } P \leq_\perp N\}.$$

Example 4.7. *We present some examples of finite approximation.*

- $\mathcal{A}(\mathsf{I}) = \{\perp, \lambda x.x\}$ *and* $\mathcal{A}(\mathsf{1}) = \{\perp, \lambda xy.x\perp, \lambda xy.xy\}$.
- $\mathcal{A}(\Omega) = \mathcal{A}(\mathsf{YI}) = \{\perp\}$*, whence* $\mathcal{A}(\lambda x.x\Omega) = \{\perp, \lambda x.x\perp\}$.

The following properties are well established. See, e.g., [1].

Lemma 4.8. *The following statements hold.*

1. $M \in \Lambda_\perp$ *is in* $\beta\perp$*-normal form if and only if* $M \in \mathcal{A}$.
2. *For* $M \in \Lambda$*, the set* $\mathcal{A}(M)$ *is an* ideal *(i.e. non-empty, downward closed and directed).*

In particular, we have that $\mathcal{A}(M)$ lives in the *ideal completion* of $\mathcal{A}$. For $A \in \mathcal{A}$, we denote as $\downarrow A$ the *principal ideal* associated to A, *i.e.* $\downarrow A = \{B \in \mathcal{A} \mid B \leq_\perp A\}$.

The (syntactic) Approximation Theorem below shows that infinite Böhm trees can be recovered by taking the supremum of their finite approximants.

Theorem 4.9 (Approximation Theorem)**.** *For all* $M \in \Lambda$ *we have*

$$\mathrm{BT}(M) = \bigvee_{A \in \mathcal{A}(M)} \downarrow A$$

Such a supremum always exists by Lemma 4.82. Moreover, $\mathrm{BT}(M) = \mathrm{BT}(N) \Leftrightarrow \mathcal{A}(M) = \mathcal{A}(N)$.

5 Simple Types

Type systems are a class of trustworthy techniques that are used to obtain certificates of correct computational behaviour for programs. Types are *specifications* of program, *i.e.* a formal description of what the considered program is supposed to do. In this section, we introduce and study the most basic type system, *simply typed* λ*-calculus* (aka the Curry type system). Simply typed λ-terms enjoy good computational properties, such as *strong normalization.* However, the class of simply typed λ-terms is very restrictive and do not contain all the recursive functions.

5.1 Propositions as Types

We fix a countable set of *atomic types* At. The *Simple types over* At are defined by the following grammar:

$$\mathsf{Ty}_{\mathsf{At}} \ni A, B ::= o \in \mathsf{At} \mid A \Rightarrow B$$

the type $A \Rightarrow B$ is called a *function* or *arrow* type. Simple types correspond to formula of minimal intuitionistic logic[2], taking atoms as propositional variables and arrow types as *implication formulas.* A *context* is a sequence of formulas $A_1, \ldots, A_n$. We denote contexts with capital Greek letters $\Gamma, \Delta \ldots$ The *derivations* of minimal intuitionistic logic are labelled trees defined by induction as follows:

$$\frac{}{A_1, \ldots, A_i, \ldots, A_n \vdash A_i} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$$

$$\frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

5.2 Typing *à la* Curry / Typing *à la* Church

The typing *à la* Curry consists in associating minimal logic derivations to pure λ-terms. The derivations become then *type* derivations, *i.e.* formal specification assignments for programs.

A *type context* is a sequence of type declaration for variables $x_1 : A_1, \ldots, x_n : A_n$. We denote type contexts with capital Greek letters $\Gamma, \Delta \ldots$ The *simple type assignment* for pure λ-calculus (aka *Curry type system*) is defined by induction on the structure of λ-terms as follows:

$$\frac{}{x_1 : A_1, \ldots, x_i : A_i, \ldots, x_n : A_n \vdash x_i : A_i} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \Rightarrow B}$$

$$\frac{\Gamma \vdash M : A \Rightarrow B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

We write $\pi \triangleright \Gamma \vdash M : A$ meaning that π is a type derivation with conclusion $\Gamma \vdash M : A$.

Another way to assign simple types to terms is given by the typing *à la Church.*

$$\Lambda^{\mathsf{st}}(\mathsf{Ty}_{\mathsf{At}}) \ni M, N := x \mid \lambda x^A.M \mid MN$$

$$\frac{}{x_1 : A_1, \ldots, x_i : A_i, \ldots, x_n : A_n \vdash x_i : A_i} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A.M : A \Rightarrow B}$$

$$\frac{\Gamma \vdash M : A \Rightarrow B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

Typing *à la Church* determines a very strong relationship between terms and typing derivations: a term can be identified with its derivation. The formal statement of this fact is the following proposition.

Proposition 5.1. *Let* $M \in \Lambda^{\mathsf{st}}(\mathsf{Ty}_{\mathsf{At}})$*. If* $\pi \triangleright \Gamma \vdash M : A$ *and* $\pi' \triangleright \Gamma \vdash M : A'$ *then* $\pi = \pi'$ *and* $A = A'$*.*

Proof. By induction on the structure of M.

Let $M = x$. The result is an immediate consequence of the definitions, since the typing of the variable is univocally determined by the context.

[2]That consists of the implication fragment of intuitionistic logic.

Let $M = \lambda x^A.M'$. Then $\pi =$

$$\dfrac{\begin{array}{c}\pi' \\ \vdots \\ \Gamma, x : A \vdash M : B\end{array}}{\Gamma \vdash \lambda x^A.M' : A \Rightarrow B}$$

and $\rho =$

$$\dfrac{\begin{array}{c}\rho' \\ \vdots \\ \Gamma, x : A \vdash M : B'\end{array}}{\Gamma \vdash \lambda x^A.M' : A \Rightarrow B'}$$

We can then apply the IH and conclude.

Let $M = PQ$. Then $\pi =$

$$\dfrac{\begin{array}{c}\pi_1 \\ \vdots \\ \Gamma \vdash M : A \Rightarrow B\end{array} \quad \begin{array}{c}\pi_2 \\ \vdots \\ \Gamma \vdash N : A\end{array}}{\Gamma \vdash MN : B}$$

and $\pi' =$

$$\dfrac{\begin{array}{c}\pi'_1 \\ \vdots \\ \Gamma \vdash M : A' \Rightarrow B'\end{array} \quad \begin{array}{c}\pi'_2 \\ \vdots \\ \Gamma \vdash N : A'\end{array}}{\Gamma \vdash MN : B'}$$

by IH, since $\pi_1 \lhd M : A \Rightarrow B$ and $\pi'_1 \lhd M : A' \Rightarrow B'$ then $A \Rightarrow B = A' \Rightarrow B'$ and $\pi_1 = \pi'_1$. We can apply the same kind of reasoning to N and conclude. □

5.3 Simple Types under Reduction

The main basic properties of typing under reduction are *subject reduction* and *subject expansion*:

1. (Subject reduction) If $\Gamma \vdash M : A$ and $M \to N$ then $\Gamma \vdash N : A$.
2. (Subject expansion) If $\Gamma \vdash N : A$ and $M \to N$ then $\Gamma \vdash M : A$.

We shall now prove that the simple type system satisfies (1), while it does not satisfy (2), as shown in Example 5.4. Intersection types will instead satisfy both.

Lemma 5.2 (Substitution Lemma)**.** *If* $\Gamma, x : A, \Delta \vdash M : B$ *and* $\Gamma, \Delta \vdash N : A$ *then* $\Gamma, \Delta \vdash M\{N/x\} : B$.

Proof. By induction on the structure of M. If $M = x$ then $M\{N/x\} = N$ and $B = A$ so the result is immediate by definition.

If $M = \lambda y.M'$, By definition we have $B = C \Rightarrow D$ and

$$\dfrac{\Gamma, x : A, \Delta y : C \vdash M' : D}{\Gamma, x : A, \Delta \vdash \lambda y.M' : C \Rightarrow D}$$

By the IH we have that $\Gamma, \Delta, y : C \vdash M'\{N/x\} : D$ hence $\Gamma, \Delta \vdash \lambda y.(M'\{N/x\}) : C \Rightarrow D$. by definition $\lambda y.M'\{N/x\} = \lambda y.(M'\{N/x\})$. We can then conclude.

If $M = PQ$ then we have

$$\dfrac{\Gamma, x : A, \Delta \vdash P : C \Rightarrow B \quad \Gamma, x : A, \Delta \vdash Q : C}{\Gamma, x : A, \Delta \vdash PQ : B}$$

the result is again a direct application of the IH, since $PQ\{N/x\} = P\{N/x\}Q\{N/x\}$. □

Proposition 5.3 (Subject Reduction). *If $\Gamma \vdash M : A$ and $M \to N$ then $\Gamma \vdash N : A$.*

Proof. By induction on the structure of the reduction step $M \to N$. If $M = (\lambda x.P)Q$ and $N = P\{Q/x\}$ then the result is a corollary of the former lemma. The other cases follow directly from the IH. □

Example 5.4 (Failure of Subject expansion). *We build a counterexample to subject expansion for simple types. Let $M = w(\lambda v.v)$ with the typing $w : (A \Rightarrow A) \Rightarrow (A \Rightarrow A) \vdash w(\lambda v.v) : A \Rightarrow A$. Now consider $I = \lambda x.x$ with the typing $z : A \Rightarrow A, w : o \vdash I : A \Rightarrow A$. We have that $I\{M/z\} = I$ and so that $z : A \Rightarrow A, w : o \vdash I\{M/z\} : A \Rightarrow A$. However, we cannot type M under the same context, since w cannot have the atomic type (it's a function). Hence*

$$(\lambda z.\lambda x.x)M \to \lambda x.x$$

with $z : A \Rightarrow A, w : o \vdash I : A \Rightarrow A$, but the judgement $z : B, w : o \vdash M : B$ is not derivable for any simple type B.

5.4 Simple Types are Strongly Normalizing

We now present a proof of strong normalization for the simply typed λ-calculus exploiting Tait-Girard *reducibility techniques.* The reducibility argument depends on the definition of an appropriate semantics for types, that assigns a set of λ-terms to each type. These terms are seen as the *realizers* of the considered type. This set has to be appropriately coherent wrt λ-terms reduction (it is a *saturated* set, cfr Definition 5.5). The proof consists essentially of two steps:

1. proving an *adequacy lemma* that links typability with realizability: if a term has type A then is also a realizer of A.

2. Proving that the realizers of a type satisfy the required property (in our case, that they are strongly normalizing).

Definition 5.5. *A subset $X \subseteq \Lambda$ is* saturated *when if $\lambda\vec{x}.P\{Q/x\}\vec{Q} \in X$ then $\lambda\vec{x}.(\lambda x.P)Q\vec{Q} \in X$. We denote as $\mathsf{sat}(\wp\Lambda)$ the set of saturated subsets of Λ.*

Given $X, Y \in \wp\Lambda$ we set $X \Rightarrow Y = \{M \in \Lambda \mid MN \in Y \text{ for all } N \in X\}$. A *reducibility interpretation* is a function $\mathsf{At} \to \mathsf{sat}(\wp\Lambda)$. Given $A \in \mathsf{Ty}_{\mathsf{At}}$, we define the *set of realizers for A* by induction as follows:

$$[\![o]\!]_I = I(o) \qquad [\![A \Rightarrow B]\!]_I = [\![A]\!]_I \Rightarrow [\![B]\!]_I.$$

It is easy to check that $[\![A]\!]_I \in \mathsf{sat}(\wp\Lambda)$.

Lemma 5.6 (Adequacy). *Let I be a reducibility interpretation. Let $x_1 : A_1, \ldots, x_n : A_n \vdash M : B$ and $N_i \in [\![A_i]\!]_I$ for $i \in [n]$. Then $M\{N_1, \ldots, N_n/x_1, \ldots, x_n\} \in [\![B]\!]_{I_{SN}}$.*

Proof. By induction on the structure of M. If $M = x_j$ for some $j \in [n]$, then $M\{N_1, \ldots, N_n/x_1, \ldots, x_n\} = N_j$ and the result is given by the hypothesis that $N_j \in [\![N_j]\!]_I$. If $M = \lambda x.M'$ then $B = C \Rightarrow D$ for some simple types C, D. By definition, $M\{N_1, \ldots, N_n/x_1, \ldots, x_n\} = \lambda x.M\{N_1, \ldots, N_n/x_1, \ldots, x_n\}$. We have to prove that for all $N \in [\![C]\!]_I, M\{N_1, \ldots, N_n/x_1, \ldots, x_n\} \in [\![D]\!]_I$. By IH we have that $M\{N_1, \ldots, N_n, N/x_1, \ldots, x_n, x\} \in [\![D]\!]_I$ for all $N \in [\![C]\!]_I$. By the fact that $[\![D]\!]_I$ is saturated, we get that $(\lambda x.M\{N_1, \ldots, N_n/x_1, \ldots, x_n\})N \in [\![D]\!]_I$. We can then conclude. If $M = PQ$ there exists a simple type C such that $x_1 : A_1, \ldots, x_n : A_n \vdash P : C \Rightarrow B$ and $x_1 : A_1, \ldots, x_n : A_n \vdash Q : A$. By definition $M\{N_1, \ldots, N_n/x_1, \ldots, x_n\} = P\{N_1, \ldots, N_n/x_1, \ldots, x_n\}Q\{N_1, \ldots, N_n/x_1, \ldots, x_n\}$. By IH we have that $P\{N_1, \ldots, N_n/x_1, \ldots, x_n\} \in [\![A \Rightarrow B]\!]_I$ and $Q\{N_1, \ldots, N_n/x_1, \ldots, x_n\} \in [\![A]\!]_I$. Then, by definition of $[\![A \Rightarrow B]\!]_I$ we can conclude that $substPx_1, \ldots, x_nN_1, \ldots, N_n \in [\![B]\!]_I$. □

We set $\mathsf{SN} = \{M \in \Lambda \mid M$ is strongly normalizable $\}$ and $\mathsf{SN}_0 = \{xN_1 \dots N_k \mid N_i \in \mathsf{SN}\}$. We trivially have that $\mathsf{SN}_0 \subset \mathsf{SN}$. We define a constant function $I_{SN} : \mathsf{At} \to \wp\Lambda$ by setting $I_{SN}(o) = \mathsf{SN}$.

Lemma 5.7. *The following statements hold.*

- SN *is saturated.*
- *For all* $A \in \mathsf{Ty}_{\mathsf{At}}$ *we have* $\mathsf{SN}_0 \subseteq [\![A]\!]_{I_{SN}} \subseteq \mathsf{SN}$.

Proof.

- By definition of strong normalization.
- By induction on the structure of A.

□

Theorem 5.8. *Let* $x_1 : A_1, \dots, x_n : A_n \vdash M : B$. *Then* M *is strongly normalizing.*

Proof. By Lemma 5.7 we have that $x_i \in \mathsf{SN}_0 \subset [\![A_i]\!]_{I_{SN}}$. By Lemma 5.6 then

$$M\{x_1, \dots, x_n / x_1, \dots, x_n\} = M \in [\![B]\!]_{I_{SN}}.$$

By Lemma 5.7 again, we know that $[\![B]\!]_{I_{SN}} \subseteq \mathsf{SN}$. □

5.5 The λY-calculus

From the point of view of type theory, the simplest way to achieve an expressive enough typed language (Turing complete) is by extending the syntax with a *fixedpoint combinator*. This new calculus is called the *λY-calculus*.

$$\Lambda_Y \ni M, N := x \mid \lambda x.M \mid MN \mid YM$$

The constructor Y is called *fixedpoint combinator*. As usual, terms are considered up to renaming of bound variables.

Typing rules:

$$\frac{}{x_1 : A_1, \dots, x_i : A_i, \dots, x_n : A_n \vdash x_i : A_i} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \Rightarrow B} \qquad \frac{\Gamma \vdash M : A \Rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$\frac{\Gamma \vdash M : A \Rightarrow A}{\Gamma \vdash YM : A}$$

Reduction base cases:

$$(\lambda x.M)N \to M\{N/x\} \qquad YM \to MYM$$

Remark 5.9. *The λY-calculus is much more expressive of the simply typed λ-calculus. However, it loses strong normalization. Terms are in general not normalizable, since the fixedpoint reduction is clearly non-terminating.*

6 Intersection Types

Intersection types were first introduced by Coppo and Dezani in the late 70s [5, 3, 6] as an extension of the simply typed λ-calculus. The main intuition behind these type disciplines is to take into account the fact that a program can be typed in different ways. A term M both typable with a type A and B is then typable with the *intersection type* $A \cap B$. Intersection type theories are very powerful and are generally able to characterize *dynamic properties* of programs. A first result in this direction was the characterization of strongly normalizing terms: a term is typable in an appropriate intersection type system if and only if it is strongly normalizable (see Chapter 4 of [7]). One can achieve this kind of results because, to the contrary of what happens with most type theories, intersection types enjoy not only subject reduction, but also *subject expansion.* The expressive power of intersection types comes with a cost: typability is not decidable, being equivalent to the Turing halting problem.

Intersection types have been presented in many slightly different ways (see [2] for a survey). We shall introduce two (parametric) variants of them, that we call *(biased) representable* and *essential* intersection types. The representable intersection types consist of an extension of the Curry type system where we add a binary *monoidal product* $\cap$ (the *intersection type* constructor) with unit ω. In the essential intersection type system instead the interaction type is seen as a list of types $\langle a_1, \dots, a_k \rangle$ for $k \in \mathbb{N}$ - which stands for the *unbiased* intersection $a_1 \cap \dots \cap a_k$ - and it appears only as *premise* of an arrow type. Essential intersection types can be easily seen as a proper subsystem of (un)biased representable intersection types, but typability in the two variants is equivalent, as we shall see.

6.1 (Biased) Representable Intersection Types

We fix a *preorder of atoms* At. The *set of intersection types over* At is defined by induction as follows:

$$|\mathsf{ITy}(\mathsf{At})| \ni a, b ::= o \in \mathsf{At} \mid a \multimap b \mid a \cap b \mid \omega$$

Types of the shape $a \cap b$ are called *intersection types.* The constant ω is the *empty intersection.* We assume that $\langle |\mathsf{ITy}(\mathsf{At})|, \cap, \omega \rangle$ is a monoid, with $\cap$ as product and ω as unit. The *basic preorder of types* $\mathsf{ITy}(\mathsf{At})$ *over* At is defined as the smallest preorder on $|\mathsf{ITy}(\mathsf{At})|$ generated by the following rules:

$$\frac{o \leq_{\mathsf{At}} o'}{o \leq_{\mathsf{ITy}(\mathsf{At})} o'} \qquad \frac{b \leq_{\mathsf{ITy}(\mathsf{At})} a \qquad a' \leq_{\mathsf{ITy}(\mathsf{At})} b'}{a \multimap a' \leq_{\mathsf{ITy}(\mathsf{At})} b \multimap b'} \qquad \frac{a \leq_{\mathsf{ITy}(\mathsf{At})} a' \qquad b \leq_{\mathsf{ITy}(\mathsf{At})} b'}{a \cap b \multimap\!\leq_{\mathsf{ITy}(\mathsf{At})} a' \cap b'}$$

We can also consider additional *structural rules*:

$$\frac{}{a \leq_{\mathsf{ITy}(\mathsf{At})} a \cap a}\ \text{contr} \qquad \frac{}{a \leq_{\mathsf{ITy}(\mathsf{At})} \omega}\ \text{top} \qquad \frac{}{a \cap b \leq_{\mathsf{ITy}(\mathsf{At})} b \cap a}\ \text{sym}$$

The *linear preorder of types* $\mathsf{ITy}(\mathsf{At})^l$ is generated by adding the sym rule to $\mathsf{ITy}(\mathsf{At})$. The *affine preorder of types* $\mathsf{ITy}(\mathsf{At})^a$ is generated by adding the sym and top rules to $\mathsf{ITy}(\mathsf{At})$. The *relevant preorder of types* $\mathsf{ITy}(\mathsf{At})^r$ is generated by adding the sym and contr rules to $\mathsf{ITy}(\mathsf{At})$. The *Cartesian preorder of types* $\mathsf{ITy}(\mathsf{At})^c$ is generated by adding all the structural rules to $\mathsf{ITy}(\mathsf{At})$.

We shall now define a *parametric intersection type system* over $\mathsf{ITy}(\mathsf{At})^\spadesuit$ with $\spadesuit \in \{l, a, r, c\}$. The system is parametric over the choice of At and $\spadesuit$.

We denote *type contexts* $x_1 : a_1, \dots, x_n : a_n$ by small Greek letters $\gamma, \delta \dots$ Given type contexts $\gamma = x_1 : a_1, \dots, x_n : a_n$ and $\delta = x_1 : b_1, \dots, x_n : b_n$ we set $\gamma \cap \delta = x_1 : a_1 \cap b_1, \dots, x_n : a_n \cap b_n$. The typing rules are defined by induction as follows:

$$\frac{}{x_1 : \omega, \ldots, x_i : a_i, \ldots, x_n : \omega \vdash x_i : a_i}\ \text{var} \qquad \frac{\gamma \vdash M : a \multimap b \qquad \delta \vdash N : a}{\gamma \cap \delta \vdash MN : b}\ \text{app} \qquad \frac{\gamma, x : a \vdash M : b}{\gamma \vdash \lambda x.M : a \multimap b}\ \text{abs}$$

$$\frac{\gamma \vdash M : a \qquad \delta \vdash M : b}{\gamma \cap \delta \vdash M : a \cap b}\ \cap \qquad \frac{\gamma \vdash M : a \qquad \delta \leq \gamma}{\delta \vdash M : a}\ \leq \qquad \frac{\gamma \vdash M : a \qquad a \leq b}{\gamma \vdash M : b}\ \leq \qquad \frac{}{x_1 : \omega, \ldots, x_n : \omega \vdash M : \omega}\ \omega$$

In the case that the preorder $\mathsf{ITy}(\mathsf{At})^{\spadesuit}$ is Cartesian, we have also an *additive presentation* of the type system:

$$\frac{}{x_1 : a_1, \ldots, x_i : a_i, \ldots, x_n : a_n \vdash x_i : a_i} \qquad \frac{\gamma \vdash M : a \multimap b \qquad \gamma \vdash N : a}{\gamma \vdash MN : b} \qquad \frac{\gamma, x : a \vdash M : b}{\gamma \vdash \lambda x.M : a \multimap b}$$

$$\frac{\gamma \vdash M : a \qquad \gamma \vdash M : b}{\gamma \vdash M : a \cap b} \qquad \frac{\gamma \vdash M : a \qquad a \leq b}{\gamma \vdash M : b} \qquad \frac{}{\gamma \vdash M : \omega}$$

Remark 6.1. *The additive presentation corresponds to the type system introduced by Coppo, Dezani and Barendregt [3]. It is worth noting that the subtyping rule on contexts is now an admissible rule, thanks to the fact that Cartesian intersection types allow the duplication of contexts.*

One can also define an *unbiased* version of intersection types, where we have k-ary intersection types $a_1 \cap \cdots \cap a_k$ for all $k \in \mathbb{N}$. In that case, the unit ω gives the 0-ary intersection type. The typing rules for this presentation of intersection types are a straightforward extension of the rules we gave above. The unbiased definition is one of the ingredients of *essential* intersection types, that is the topic of the next section.

6.2 Essential Intersection Types

We fix again a *preorder of atoms* At. The *set of essential intersection types over* At is defined by induction as follows:

$$|\mathsf{ITy}(\mathsf{At})^{\spadesuit}_e| \ni a, b ::= o \in \mathsf{At} \mid \langle a_1, \ldots, a_k \rangle \multimap b$$

with $k \in \mathbb{N}$.

The *preorder of essential intersection types* $\mathsf{ITy}(\mathsf{At})^{\spadesuit}_e$ *over* At is defined as the smallest preorder on $|\mathsf{ITy}(\mathsf{At})^{\spadesuit}_e|$ generated by the following rules:

$$\frac{o \leq_{\mathsf{At}} o'}{o \leq_{\spadesuit} o'} \qquad \frac{\vec{b} \leq_{\spadesuit} \vec{a} \qquad a' \leq_{\spadesuit} b'}{\vec{a} \multimap a' \leq_{\spadesuit} \vec{b} \multimap b'} \qquad \frac{\alpha \in \mathcal{O}^{\spadesuit}([m], [n]) \qquad a_{\alpha(i)} \leq_{\spadesuit} b_i}{\langle a_1, \ldots, a_n \rangle \leq_{\spadesuit} \langle b_1, \ldots, b_m \rangle}$$

A *(essential) type context* consists of a sequence of variable type declarations $x_1 : \vec{a}_1, \ldots, x_n : \vec{a}_n$. Given type contexts $\gamma = x_1 : \vec{a}_1, \ldots, x_n : \vec{a}_n$ and $\delta = x_1 : \vec{b}_1, \ldots, x_n : \vec{b}_n$ we set $\gamma \otimes \delta = x_1 : \vec{a}_1 :: \vec{b}_1, \ldots, x_n : \vec{a}_n :: \vec{b}_n$.

The typing rules:

$$\frac{\vec{a}_i \leq_{\spadesuit} \langle a \rangle \qquad \vec{a}_j \leq_{\spadesuit} \langle\rangle,\ \text{for } j, i \in [n], j \neq i}{x_1 : \vec{a}_1, \ldots, x_i : \vec{a}_i, \ldots, x_n : \vec{a}_n \vdash x_i : a} \qquad \frac{\gamma, x : \vec{a} \vdash M : b}{\gamma \vdash \lambda x.M : \vec{a} \multimap b}$$

$$\frac{\gamma_0 \vdash M : \langle a_1, \ldots, a_k \rangle \multimap b \qquad (\gamma_i \vdash N : a_i)_{i=1}^k \qquad \delta \leq_{\spadesuit} \bigotimes_{j=0}^k \gamma_j}{\delta \vdash MN : b}$$

$$\frac{\vec{a}_i \leq_c \langle a \rangle}{x_1 : \vec{a}_1, \ldots, x_i : \vec{a}_i, \ldots, x_n : \vec{a}_n \vdash x_i : a} \qquad \frac{\gamma, x : \vec{a} \vdash M : b}{\gamma \vdash \lambda x.M : \vec{a} \multimap b}$$

$$\frac{\gamma \vdash M : \langle a_1, \ldots, a_k \rangle \multimap b \qquad (\gamma \vdash N : a_i)_{i=1}^{k}}{\gamma \vdash MN : b}$$

Figure 1: The Cartesian type system.

$$\frac{a' \leq_l a}{x_1 : \langle\rangle, \ldots, x_i : \langle a' \rangle, \ldots, x_n : \langle\rangle \vdash x_i : a} \qquad \frac{\gamma, x : \vec{a} \vdash M : b}{\gamma \vdash \lambda x.M : \vec{a} \multimap b}$$

$$\frac{\gamma_0 \vdash M : \langle a_1, \ldots, a_k \rangle \multimap b \qquad (\gamma_i \vdash N : a_i)_{i=1}^{k} \qquad \delta \leq_l \bigotimes_{j=0}^{k} \gamma_j}{\delta \vdash MN : b}$$

Figure 2: The linear type system.

It is worth noting that in the system presented above there is a basic asymmetry between the typing contexts and the type of a term. The type declaration of variables in context is indeed given by types lists, while the type of the term can never be a list.

Again, the Cartesian case has an additive presentation, as shown in Figure 1.

Remark 6.2. *We shall constantly keep implicit the subtyping choice in a rule in the case we are taking reflexivity. So, for instance, we shall write*

$$\frac{}{x : \langle a \rangle \vdash x : a} \qquad \frac{\gamma_0 \vdash M : \langle a_1, \ldots, a_k \rangle \multimap b \qquad (\gamma_i \vdash N : a_i)_{i=1}^{k}}{\bigotimes \gamma_j \vdash MN : b}$$

instead of

$$\frac{\langle a \rangle \leq \langle a \rangle}{x : \langle a \rangle \vdash x : a} \qquad \frac{\gamma_0 \vdash M : \langle a_1, \ldots, a_k \rangle \multimap b \qquad (\gamma_i \vdash N : a_i)_{i=1}^{k} \qquad \bigotimes \gamma_j \leq \bigotimes \gamma_j}{\bigotimes \gamma_j \vdash MN : b}$$

We define by induction an embedding of essential intersection types into representable intersection types $T : |\mathsf{ITy}(\mathsf{At})_e^{\spadesuit}| \to |\mathsf{ITy}(\mathsf{At})^{\spadesuit}|$:

$$T(o) = o \qquad T(\langle a_1, \ldots, a_k \rangle \multimap a) = \begin{cases} T(a_1) \cap \cdots \cap T(a_k) \multimap T(a) & \text{if } k > 0. \\ \omega \multimap T(a) & \text{otherwise.} \end{cases}$$

The former function can be extended to a monotonic function between the two preorders in a straightforward way.

Theorem 6.3. *M is typable in the representable intersection type system if and only if it is typable in the essential intersection type system.*

Proof. ($\Rightarrow$) If M is typable then there exists a type derivation $\pi \triangleright \gamma \vdash_b M : a$. The result is a straightforward induction on the structure of π.

($\Leftarrow$) It follows by observing that the embedding T preserves typability. $\square$

Example 6.4. *We present some examples of type derivations for essential intersection types.*

1. *One one the main differences between simple types and intersection types is that the latter can be used to type auto-applications such as xx. This is due to the* polymorphic nature *of intersection types, that can account for different computational behaviours of terms. A typing of $\lambda x.xx$:*

$$\dfrac{\dfrac{}{x : \langle\langle a\rangle \multimap a\rangle \vdash : x\langle a\rangle \multimap a} \qquad \dfrac{}{x : \langle a\rangle \vdash x : a}}{\dfrac{x : \langle\langle a\rangle \multimap a, a\rangle \vdash xx : a}{\vdash \lambda x.xx : \langle\langle a\rangle \multimap a, a\rangle \multimap a}}$$

2. *Typability with intersection types is much more expressive than the one with simple types. For instance, we can type* head-normal forms *that are not simply typable:*

$$\dfrac{\dfrac{}{x : \langle\langle\rangle \multimap a\rangle \vdash x : \langle\rangle \multimap a}}{x : \langle\langle\rangle \multimap a\rangle \vdash x\Omega : a}$$

   We shall see (Section 6.3) that intersection types and the head-reduction are indeed deeply connected. Typability in intersection type systems is equivalent *to head-normalization.*

3. *Relevant and irrelevant intersection types differ on the typing of weakening. While in relevant intersection type systems variables that do not appear in the body of the term must be typed via the empty intersection $\langle\rangle$, in the affine and Cartesian cases they can be typed with arbitrary types. A very basic and interesting example is the following:*

$$\dfrac{\dfrac{}{z : \langle\rangle, x : \langle a\rangle \vdash x : a}}{x : \langle a\rangle \vdash \lambda z.x : \langle\rangle \multimap a} \qquad \dfrac{\dfrac{}{z : \vec{a}, x : \langle a\rangle \vdash x : a}}{x : \langle a\rangle : \lambda z.x : \vec{a} \multimap a}$$

   As we shall see (Section 6.3) the irrelevant intersection types can characterize strong normalization via positive typing*, i.e. strong normalizing terms are typable with types where the empty intersection do not appear. This is not possible for relevant intersection types, as the example of $\lambda z.x$ shows.*

Remark 6.5. *Intersection types should not be confused with product types. Within simple types, the product is the type of a new term constructor:* pairing*. Intersection types instead determine a kind of* finite polymorphism*, encoding possible different computational behaviours that a program can have.*

(Connection with Linear Logic) If the product type corresponds to pairing, intersection types morally correspond to a 'bang' constructor, $!M$. From this point of view, it's clear why we can use just essential *intersection types within the framework of pure λ-calculus. The exponential modality $!$ is indeed implicit and it appears only in the term application. In order to be more precise, lets consider the following extension of λ-terms syntax:*

$$\Lambda_! \ni M, N ::= x \mid \lambda x.M \mid MN \mid !M$$

Thanks to this new syntax, we can make (unbiased) representable intersection types into a syntax-directed system. Consider indeed the following typing rule:

$$\dfrac{\gamma_1 \vdash M : a_1 \ldots \gamma_k \vdash M : a_k \qquad \delta \leq \bigotimes \gamma_i}{\delta \vdash !M : \langle a_1, \ldots, a_k\rangle}$$

This rules corresponds to the unbiased version of the standard introduction rule of the intersection type. However, the introduction of the intersection type now depends on the introduction of the term constructor $!$.

6.3 Intersection Types Under Reduction

We now study the relationship between intersection types and dynamic properties of λ-terms.

6.3.1 Subject Reduction and Expansion

As already mentioned, intersection types satisfy *both* subject reduction and subject expansion.

Lemma 6.6 (Subtyping is admissible)**.** *The following statements hold.*

1. *If $\gamma \vdash M : a$ and $\delta \leq_\spadesuit \gamma$ then $\delta \vdash M : a$.*
2. *If $\gamma \vdash M : a$ and $a \leq_\spadesuit b$ then $\gamma \vdash M : b$.*

Lemma 6.7 (Weakening)**.** *Let $x_1 : \vec{a}_1, \dots, x_n : \vec{a}_n \vdash M : a$ and $\vec{b}_1 \leq_\spadesuit \langle\rangle, \dots, \vec{b}_n \leq_\spadesuit \langle\rangle$. Then $x_1 : \vec{a}_1 :: \vec{b}_1, \dots, x_n : \vec{a}_n :: \vec{b}_n \vdash M : a$.*

Proof. By induction on the structure of $\pi \triangleright x_1 : \vec{a}_1, \dots, x_n : \vec{a}_n \vdash M : a$. If $M = x_i$ then we have $\pi =$

$$\frac{\vec{a}_i \leq_\spadesuit \langle a\rangle \qquad \vec{a}_j \leq_\spadesuit \langle\rangle, \text{ for } j, i \in [n], j \neq i}{x_1 : \vec{a}_1, \dots, x_i : \vec{a}_i, \dots, x_n : \vec{a}_n \vdash x_i : a}$$

then by compatibility of the preorder of types wrt list concatenation we have $\vec{a}_j :: \vec{b}_j \leq \langle\rangle :: \langle\rangle = \langle\rangle$ for $j \neq i$ and $\vec{a}_i :: \vec{b}_i \leq \langle a\rangle :: \langle\rangle = \langle a\rangle$. We can then conclude. If $M = \lambda x.M'$ then $\pi =$

$$\frac{\begin{array}{c}\pi' \\ \vdots \\ x_1 : \vec{a}_1, \dots, x_n : \vec{a}_n, x : \vec{a} \vdash M' : a\end{array}}{x_1 : \vec{a}_1, \dots, x_n : \vec{a}_n \vdash \lambda x.M' : \vec{a} \multimap a}$$

by IH we have $x_1 : \vec{a}_1 :: \vec{b}_1, \dots, x_n : \vec{a}_n :: \vec{b}_n, x : \vec{a} :: \langle\rangle \vdash M' : a$. Then we can conclude that $x_1 : \vec{a}_1 :: \vec{b}_1, \dots, x_n : \vec{a}_n :: \vec{b}_n \vdash \lambda x.M' : \vec{a} \multimap a$. If $M = PQ$ then $\pi =$ □

Proposition 6.8 ((De)substitution)**.** *The following statements hold.*

1. *If $\gamma_0, x : \langle a_1, \dots, a_k\rangle \vdash M : b, \gamma_i \vdash N : a_i$ for $1 \leq i \leq k$ and $\delta \leq_\spadesuit \bigotimes \gamma_j$ then $\delta \vdash M\{N/x\} : b$.*
2. *If $\delta \vdash M\{N/x\} : b$ then there exist an intersection type $\langle a_1, \dots, a_k\rangle$ and type contexts $\gamma_0, \dots, \gamma_k$ such that $\delta \leq_\spadesuit \bigotimes \gamma_j, \gamma_0, x : \langle a_1, \dots, a_k\rangle \vdash M : b$ and $\gamma_i \vdash N : a_i$ for $1 \leq i \leq k$.*

Proof. 1. By induction on the structure of M, exploiting Lemma 6.6.

2. By induction on the structure of M, exploiting Lemma 6.6.

□

Theorem 6.9 (Subject Reduction/Expansion)**.** *The following statements hold.*

1. *If $M \to_\beta N$ and $\gamma \vdash M : a$ then $\gamma \vdash N : a$.*
2. *If $M \to_\beta N$ and $\gamma \vdash N : a$ then $\gamma \vdash M : a$.*

Proof. 1. By induction on the structure of the reduction step $M \to_\beta N$. The base case where $M = (\lambda x.P)Q$ and $N = P\{Q/x\}$ is a direct corollary of Lemma 1. The other cases are immediate consequence of the IH.

2. By induction on the structure of the reduction step $M \to_\beta N$. The base case where $M = (\lambda x.P)Q$ and $N = P\{Q/x\}$ is a direct corollary of Lemma 2. The other cases are immediate consequence of the IH.

$\square$

6.4 Normalization Theorems

In this section we present reducibility proofs of several normalization thoerems for intersection types. In particular, we shall prove that, under appropriate assumptions, intersection types characerize head-normalization, β-normalization and strong normalization.

6.4.1 Typing of (Head) Normal Forms

Lemma 6.10. *Let $M \in \Lambda$ be a head-normal form. Then M is typable in the intersection type system $\mathcal{R}$.*

Proof. We have that $M = \lambda x_1 \dots \lambda x_m.xQ_1 \cdots Q_n$. We prove it for $xQ_1 \cdots Q_n$, choosing as list of variables $\vec{y} = \vec{x} \oplus \langle x_1, \dots, x_m\rangle = \langle y_1, \dots, y_k\rangle$ where $k = m + \mathsf{len}(\vec{x})$, the extension being immediate. Let $b = \langle\rangle \multimap \cdots \multimap \langle\rangle \multimap a$ for a arbitary type. It is enough to take the following type derivation $\pi =$

$$\dfrac{\overline{y_1 : \langle\rangle, \dots, x : \langle b\rangle, \dots, y_k : 1_{\langle\rangle} \vdash x : \langle\rangle \multimap \cdots \multimap \langle\rangle \multimap a}}{y_1 : \langle\rangle, \dots, x : \langle b\rangle, \dots, y_k : \langle\rangle \vdash xQ_1 \cdots Q_n : a}$$

$\square$

Corollary 6.11. *Let $M \in \Lambda$. If M is head-normalizable then M is typable in system R.*

Proof. Corollary of the former lemma and Theorem **??**. $\square$

Definition 6.12. *We inductively define two subsets Pos, Neg of $|\mathsf{ITy}(\mathsf{At})|$:*

- $|\mathsf{At}| \subset Pos$ *and* $|\mathsf{At}| \subset Neg$;
- *if $\vec{a} \in Neg^!$ and $a \in Pos$ then $\vec{a} \multimap a \in Pos$.*
- *if $\vec{a} \in Pos^!$ such that $\vec{a} \neq \langle\rangle$ and $a \in Neg$ then $\vec{a} \multimap a \in Neg$.*

We remark that the two considered subset defines two subpreorders of $\mathsf{ITy}(\mathsf{At})$ in the natural way. If $a \in Pos$ (resp. $a \in Neg$) we say that a is *positive* (resp. *negative*). For a type context γ, we say that it is *positive* (resp. *negative*) when all its elements are.

We also define another subset $|\mathsf{ITy}(\mathsf{At})^+| \subseteq |\mathsf{ITy}(\mathsf{At})|$ as the smallest set generated by the following grammar:

$$o \in \mathsf{At} \mid \langle a_0, \dots, a_k\rangle \Rightarrow a$$

hence if $\vec{a} \multimap a \in |\mathsf{ITy}(\mathsf{At}){+}|$ then $\vec{a} \neq \langle\rangle$. Clearly also $|\mathsf{ITy}(\mathsf{At})|^+$ defines a subpreorder of $\mathsf{ITy}(\mathsf{At})$ in the natural way.

Lemma 6.13. *Let $M \in \Lambda$ be a β-normal form. Then $\gamma \vdash M : a$ for some negative context γ and positive type a.*

Proof. By induction on the size of $M = \lambda x_1.\dots\lambda x_m.xQ_1\dots Q_n$. We set $\vec{y} = \vec{x} \oplus \langle x_1, \dots, x_m\rangle$. We prove the result for $M' = xQ_1 \dots Q_n$, the extension being immediate. By IH we have that $\gamma_i \vdash Q_i : a_i$ for some $\gamma_i \in S(Neg)^{\mathsf{len}(\vec{y})}, a_i \in Pos$ for $i \in [n]$. Consider the type $b = \langle a_1\rangle \multimap \dots \multimap \langle a_n\rangle \Rightarrow o$. Since $a_1, \dots, a_n \in Pos$ and $o \in Neg$ we have that $b \in Neg$.

Let $\gamma_0 = \langle\rangle, \dots \langle b\rangle, \dots \langle\rangle$. Then we have by definition $\bigotimes_{j=0}^n \gamma_j \vdash M : o$. □

Corollary 6.14. *Let $M \in \Lambda$. If M is β-normalizable then*

$$([\![M]\!]_{\vec{x}})_{|Pos} \neq \emptyset_{Pos,(S(Neg))^{\mathsf{len}(\vec{x})}}.$$

Proof. Corollary of the former lemma and Theorem **??**. □

Lemma 6.15. *Let $M \in \Lambda$ be a β-normal form and S be an irrelevant resource monad. Then*

$$([\![M]\!]_{\vec{x}})_{|D^+} \neq \emptyset_{D^+,(S(D^+))^{\mathsf{len}(\vec{x})}}.$$

Proof. By induction on the size of $M = \lambda x_1.\dots\lambda x_m.xQ_1\dots Q_n$. We set $\vec{y} = \vec{x} \oplus \langle x_1, \dots, x_m\rangle$. We prove the result for $M' = xQ_1 \dots Q_n$, the extension being immediate. If $n = 0$ we use the irrelevancy of the resource monad and we type x as follows

$$\frac{\diamondsuit_{\vec{a}_1}, \dots 1_a, \dots, \diamondsuit_{\vec{a}_{\mathsf{len}(\vec{y})}}}{y_1 : \vec{a}_1, \dots, x : \langle a\rangle, \dots, y_{\mathsf{len}(\vec{y})} \vdash x : a}$$

and by Theorem **??** we conclude. If $n \neq 0$ then the proof follows the same pattern as the one of Lemma 6.13. □

Corollary 6.16. *Let $M \in \Lambda$ and S be an irrelevant resource monad. If M is β-normalizable then*

$$([\![M]\!]_{\vec{x}})_{|D^+} \neq \emptyset_{D^+,(S(D^+))^{\mathsf{len}(\vec{x})}}.$$

Proof. Corollary of the former lemma and Theorem **??**. □

6.4.2 Reducibility Approach

A *reducibility interpretation* is a monotonic function $I : \mathsf{At} \to \langle \mathsf{sat}(\wp\Lambda)\rangle$, where we recall that the latter is the poset of *saturated* subsets (Definition 5.5) of Λ, ordered by inclusion.

Given a type $a \in \mathsf{ITy}(\mathsf{At})_e^{\spadesuit}$, we define the *set of I-realizers of a* by induction as follows:

$$[\![o]\!]_I = I(o) \qquad [\![\langle a_1, \dots, a_k\rangle]\!]_I = \begin{cases} \bigcap_{i=1}^k [\![a_i]\!]_I & \text{if } k > 0. \\ \Lambda & \text{otherwise.} \end{cases}$$

$$[\![\vec{a} \multimap a]\!]_I = [\![\vec{a}]\!]_I \Rightarrow [\![a]\!]_I$$

where we recall that for $X, Y \subseteq \Lambda, X \Rightarrow Y = \{M \in \Lambda \mid \text{ for all } N \in X, MN \in Y\}$. It is easy to see that $[\![a]\!]_I$ is saturated.

Lemma 6.17 (Subtyping)**.** *If $a \leq a'$ then $[\![a]\!]_I \subseteq [\![a']\!]_I$.*

Proof. By induction on the structure of a. The base case derives by the fact that I is monotonic. □

$$\frac{}{x_1 : [], \ldots, x_i : [a], \ldots, x_n : [] \vdash x_i : a} \qquad \frac{\gamma, x : \overline{a} \vdash M : b}{\gamma \vdash \lambda x.M : \overline{a} \multimap b}$$

$$\frac{\gamma_0 \vdash M : [a_1, \ldots, a_k] \multimap b \qquad (\gamma_i \vdash N : a_i)_{i=1}^k}{\sum_{j=0}^k \gamma_j \vdash MN : b}$$

Figure 3: Gardner-De Carvalho non-idempotent intersection type system

$$\frac{\tilde{a}_i \leq \{a\}}{x_1 : \tilde{a}_1, \ldots, x_i : \tilde{a}_i, \ldots, x_n : \tilde{a}_n \vdash x_i : a} \qquad \frac{\gamma, x : \tilde{a} \vdash M : b}{\gamma \vdash \lambda x.M : \tilde{a} \multimap a}$$

$$\frac{\gamma \vdash M : \{a_1, \ldots, a_k\} \multimap b \qquad (\gamma \vdash N : a_i)_{i=1}^k}{\gamma \vdash MN : b}$$

Figure 4: Ehrhard's presentation of the idempotent intersection type system.

6.4.3 Combinatorial Proofs of Normalization

In the case of *linear* and *affine* intersection types it is possible to give alternative proofs of head and β-normalization, that do not rely on impredicative tecniques as instead was the case for reducibility arguments. The proofs depend on the fact that the size of type derivations decreases under leftmost reduction. For the rest of the section, we write just R to denote the system $R^\clubsuit$ for $\clubsuit \in \{a, l\}$.

Theorem 6.18 (Affine Strong Normalization). *If $\gamma \vdash_a^+ M : a$ then M is strongly normalizing.*

6.5 From Essential Intersection Types to Multi-Types and Set-Types

We introduced intersection types as a preorder, exploiting several monoid constructions over preordered sets. However, it is common practice among researchers in the field to collapse the preorder into a partial order, *via* the canonical quotient. This process leads to particular presentations of the intersection type, based on datatypes other than lists.

Consider $\mathsf{ITy}(\mathsf{At})_e^\clubsuit$ for $\clubsuit \in \{l, a, r\}$. We define an equivalence relation over it as the smallest one generated by the following rule: $a \sim a'$ when $a \cong_\clubsuit a'$. An explicit presentation of this quotient exploits *multisets*:

$$\mathsf{ITy}(\mathsf{At})_e^\clubsuit / \sim \ni a, b ::= o \in \mathsf{At} \mid [a_1, \ldots, a_k] \multimap b$$

Remark 6.19. *It is worth noting that in the linear case, the preorder rule is slightly redundant and could be replaced by*

$$\frac{a_i \leq b_i}{[a_1, \ldots, a_k] \leq [b_1, \ldots, b_k]}$$

If we restrict to the linear case and we take a set as atomic preorder At, then $\mathsf{ITy}(\mathsf{At})_e^\clubsuit / \sim$ also collapses into a set. This particular case corresponds to the well-known Gardner-De Carvalho *non-idempotent intersection type system*, see Figure 3.

$$\frac{a \in \tilde{a}_i}{x_1 : \tilde{a}_1, \dots, x_i : \tilde{a}_i, \dots, x_n : \tilde{a}_n \vdash x_i : a} \qquad \frac{\gamma, x : \tilde{a} \vdash M : b}{\gamma \vdash \lambda x.M : \tilde{a} \multimap a}$$

$$\frac{\gamma \vdash M : \{a_1, \dots, a_k\} \multimap b \qquad (\gamma \vdash N : a_i)_{i=1}^{k}}{\gamma \vdash MN : b}$$

Figure 5: Krivine's presentation of the idempotent intersection type system.

6.6 From Simple to Intersection Types

We define an embedding $\mathsf{ST} : \mathsf{ST}(\mathsf{At}) \to |\mathsf{ITy}(\mathsf{At})|$ of simple types into (essential) intersection types by induction as follows:

$$\mathsf{ST}(o) = o \qquad \mathsf{ST}(A \Rightarrow B) = \langle \mathsf{ST}(A) \rangle \multimap \mathsf{ST}(B).$$

Theorem 6.20. *If $x_1 A_1, \dots, x_n : A_n \vdash M : A$ then $x_1 : \langle \mathsf{ST}(A_1) \rangle, \dots, x_n : \langle \mathsf{ST}(A_n) \rangle \vdash_c M : \mathsf{ST}(a)$.*

Proof. By induction on the structure of $\pi \triangleright \Gamma \vdash M : A$. □

Remark 6.21. *It is worth noting that Theorem 6.20, as it is stated, works only for Cartesian intersection types. Indeed, it is easy to find counterexamples in the other cases. Consider $f : A \Rightarrow A \Rightarrow B, x : A \vdash fxx : B$. We have $\mathsf{ST}(A \Rightarrow A \Rightarrow B) = \langle \mathsf{ST}(A) \rangle \multimap \langle \mathsf{ST}(A) \rangle \multimap \mathsf{ST}(B)$. However, in the linear an affine cases the type context is forced to be $f : \langle \langle \mathsf{ST}(A) \rangle \multimap \langle \mathsf{ST}(A) \rangle \multimap \mathsf{ST}(B) \rangle, x : \langle \mathsf{ST}(A), \mathsf{ST}(A) \rangle$, since contraction is not allowed. In the relevant case instead, the counterexample is given by any weakening, such as $y : B, x : A \vdash x : A$. However, typability with simple types clearly implies typability with intersection types in all the cases, as an indirect consequence of Theorems .*

References

[1] Roberto M. Amadio and Pierre-Louis Curien. *Domains and Lambda-calculi.* New York, NY, USA: Cambridge University Press, 1998. ISBN: 0-521-62277-8.

[2] Steffen Van Bakel. "Strict Intersection Types for the Lambda Calculus". In: 43.3 (Apr. 2011). DOI: 10.1145/1922649.1922657.

[3] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. "A Filter Lambda Model and the Completeness of Type Assignment". In: *The Journal of Symbolic Logic* 48.4 (1983), pp. 931–940. ISSN: 00224812. (Visited on 08/03/2022).

[4] Henk P. Barendregt. *The lambda-calculus, its syntax and semantics.* revised. Studies in Logic and the Foundations of Mathematics 103. 1984.

[5] Mario Coppo and Mariangiola Dezani-Ciancaglini. "A new type-assignment for lambda terms". In: *Archiv für Mathematische Logik und Grundlagenforschung.* 1978, pp. 139–156.

[6] Mario Coppo, Mariangiola Dezani-Ciancaglini, Furio Honsell, and Giuseppe Longo. "Extended Type Structures and Filter Lambda Models". In: *Logic Colloquium '82.* Ed. by G. Lolli, G. Longo, and A. Marcja. Vol. 112. Studies in Logic and the Foundations of Mathematics. Elsevier, 1984, pp. 241–262. DOI: https://doi.org/10.1016/S0049-237X(08)71819-6.

[7] Jean-Louis Krivine. *Lambda-calculus, types and models.* Ellis Horwood series in computers and their applications. 1993.

[8] Stefania Lusin and Antonino Salibra. “The Lattice of Lambda Theories”. In: *J. Log. Comput.* 14.3 (2004), pp. 373–394.

[9] Christopher P. Wadsworth. “The Relation Between Computational and Denotational Properties for Scott’s $\mathcal{D}_\infty$-Models of the Lambda-Calculus”. In: *SIAM J. Comput.* 5.3 (1976), pp. 488–521. URL: `https://doi.org/10.1137/0205036`.