Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 398 (2008) 95–113

Theoretical Computer Science

www.elsevier.com/locate/tcs

# A typed lambda calculus with intersection types[☆]

Viviana Bono[a,*], Betti Venneri[b], Lorenzo Bettini[a]

[a] Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149 Torino, Italy
[b] Dipartimento di Sistemi e Informatica, Università di Firenze, Viale Morgagni, 65, 50134 Firenze, Italy

**Abstract**

Intersection types are well known to type theorists mainly for two reasons. Firstly, they type all and only the strongly normalizable lambda terms. Secondly, the intersection type operator is a meta-level operator, that is, there is no direct logical counterpart in the Curry–Howard isomorphism sense. In particular, its meta-level nature implies that it does not correspond to the intuitionistic conjunction.

The intersection type system is naturally a type inference system (system à la Curry), but the meta-level nature of the intersection operator does not allow to easily design an equivalent typed system (system à la Church). There are many proposals in the literature to design such systems, but none of them gives an entirely satisfactory answer to the problem. In this paper, we will review the main results in the literature both on the logical interpretation of intersection types and on proposed typed lambda calculi.

The core of this paper is a new proposal for a true intersection typed lambda calculus, without any meta-level notion. Namely, any typable term (in the intersection type inference) has a corresponding typed term (which is the same as the untyped term by erasing the type decorations and the typed term constructors) with the same type, and vice versa.

The main idea is to introduce a *relevant* parallel term constructor which corresponds to the intersection type constructor, in such a way that terms in parallel share the same resources, that is, the same context of free typed variables. Three rules allow us to generate all typed terms. The first two rules, Application and Lambda-abstraction, are performed on all the components of a parallel term in a synchronized way. Finally, via the third rule of Local Renaming, once a free typed variable is bounded by lambda-abstraction, each of the terms in parallel can do its local renaming, with type refinement, of that particular resource.
© 2008 Elsevier B.V. All rights reserved.

*Keywords:* Intersection types; Lambda calculus; Type inference; Church style; Curry style; Parallelism; Shared resources

## 1. Introduction

The history of intersection types starts in the late seventies: they were introduced in [6,7] as a generalization of Curry's type inference system, in order to characterize neatly a larger class of terms. The main idea is the introduction of a new type-forming operator, the intersection $\wedge$, whose introduction and elimination rules are:

$$\frac{\Gamma \Vdash M : \tau_1 \qquad \Gamma \Vdash M : \tau_2}{\Gamma \Vdash M : \tau_1 \wedge \tau_2}\,(\wedge I) \qquad \frac{\Gamma \Vdash M : \tau_1 \wedge \tau_2}{\Gamma \Vdash M : \tau_i \qquad i = 1, 2}\,(\wedge E).$$

[☆] This work has been partially supported by the EU Project Software Engineering for Service-Oriented Overlay Computers (SENSORIA, contract IST-3-016004-IP-09) and by the MIUR project EOS DUE.
[*] Corresponding author.
*E-mail address:* viviana.bono@di.unito.it (V. Bono).

0304-3975/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.01.046

As it is mentioned in [36], intersection types provide type polymorphism by listing type instances, differing from the more widely used ∀-quantified types [13,25], which provide type polymorphism by giving a type scheme that can be instantiated into various type instances via different substitutions of types for quantified type variables.

The system with intersection ITD (Intersection Type Discipline, also known as Intersection Type Assignment $\mathrm{TA}_\wedge$, or as Intersection Type Inference) turns out to be significantly powerful, since it allows the typing of exactly the strongly normalizable terms (terminating programs). Thus, for instance, it is more powerful than the ∀ polymorphism with respect to the set of typable terms (see, for instance, [31,17]).

Another remarkable side of the intersection type operator is its proof-functional nature. In fact, if we look at the intersection type discipline from the point of view of the formulae-as-types approach (also known as the Curry–Howard isomorphism, [16]) there is a surprise.

The Curry–Howard isomorphism is the mapping of constructive proofs of logical formulae into programs (lambda terms or combinators) of related type and conversely. Thus a logical meaning for the type constructor is provided, while logical systems can be seen in a computational way. As far as the simple types of Curry system are concerned, the well-known analogy with the implicational fragment of intuitionistic propositional logic $\mathrm{L}_\to$ is quite natural. Roughly speaking, an arrow type of the shape $\alpha \to \beta$ is paralleled by the implicational formula $A \to B$, where the elimination and the introduction rules for the arrow in types correspond, respectively, to the introduction and elimination rule for the implication in natural deduction, and reduction of typable terms to normal forms corresponds to normalization of proofs. The simple type discipline has been extended to stronger systems, still prompted by the Curry–Howard isomorphism. In particular, second- and higher-order logics can be dealt with by introducing connectives and quantifiers (of suitable order) on types, thus generating Girard's system F and $\mathrm{F}_\omega$, [14] (see [30] for a thorough account on Curry–Howard isomorphism).

For ITD, however, the question "what is the corresponding logical system like?" has not yet been answered satisfactorily. Let us consider where the difficulty lies. Since ITD extends the Curry system by adding a new type constructor with corresponding rules, it is reasonable to look for a logic $\mathrm{L}_\wedge$, extending $\mathrm{L}_\to$ both in the language and in the set of deduction rules, such that the above mentioned isomorphism holds between $\mathrm{L}_\wedge$ and ITD.

Difficulties arise from the specific shape of the $(\wedge I)$ rule, saying that a term has (a deduction proves) a type (a formula) $\sigma \wedge \tau$ if and only if this term has (the *same* deduction proves) both $\sigma$ and $\tau$. Therefore the first candidate to correspond to intersection seems to be a restricted form of the usual intuitionistic propositional conjunction &: in other words, the provability of a conjunctive formula $A\&B$ must require that both conjuncts are provable by proofs with the *same* structure. In such a system, the &-introduction rule would be constrained by a global meta-linguistic condition of applicability involving the shape of the whole subderivations.

These features led some authors to investigate intersection as a proof-functional (as opposed to truth-functional) operator, in the context of "untyped terms as *realizers* of logical formulae". Lopez-Escobar first referred to ∧ as "... the first... connective which is truly proof-functional", [19]. Following that approach, in [20] and [1], a first-order logic was defined to derive predicate formulae such as $R_A[M]$, meaning "the lambda term $M$ realizes the propositional formula $A$". Actually, in this logic no specific rule is given to represent the ∧-derivability; the predicate $R_{A\wedge B}[M]$ is proved by the proof of the predicate $R_A[M]\&R_B[M]$, which is defined as equivalent to the former one since the two subjects of the predicates connected by & are equal (where & is the usual conjunction).

A question arises: what is the main consequence of dealing with a proof-functional operator from the point of view of types? The main consequence is that the intersection type system is naturally a type inference system (system à la Curry), but the meta-level nature of the intersection operator does not allow to easily design an equivalent (explicitly) typed system (system à la Church). Note that, in fact, no syntax is introduced in the term in order to match the intersection introduction in the $(\wedge I)$ rule, which implies that in ITD there are terms that do not encode deductions, therefore an immediate issue which arises is how to make a type-annotated variant of the system. There are many proposals in the literature to design such a typed system, which we will discuss later, but none of them gives an entirely satisfactory answer to the problem addressed in the present paper.

The core of this paper is a new proposal for a true intersection typed lambda calculus, without any meta-level notion. Namely, any typable term (in the intersection type inference) has a corresponding typed term (which is the same as the untyped term by erasing the type decorations and the typed term constructors) with the same type, and vice versa.

Let us discuss what is needed to get to a typed intersection system. First of all, it follows from the discussion on the logical meaning of intersection that it is necessary to design a system where there is no intersection introduction as such. In order to do so, we shall define a system proved to be equivalent to ITD, in which no rule involves any proof-functional condition for the rule's applicability, by containing relations on subderivations. Then, we must deal with the lambda-abstraction operation carefully: intuitively, when an intersection typed variable is bounded, we must have a mechanism to record in the term the different uses we make of such a *resource* in the term itself (that is, only some parts of its type may be exploited for any occurrence of the variable). In fact, Venneri, [34,10], succeeded in completely removing the $(\wedge I)$ rule from a type system with intersection types, but this was for combinatory logic (that obviously does not have any form of lambda-abstraction), rather than for the lambda calculus, and that particular approach seems unlikely to be transferable to the lambda calculus. But from this approach we learn something: the intersection operator has been shown there to correspond to a mix of the intuitionistic conjunction of different instances of the same theorem and the application of theorems on conjunction in a *relevant* logic [21].

Intuitively, our main idea is then to introduce a *relevant* parallel term constructor representing the intersection, which permits performing an intersection introduction only at the very start of a type derivation via the Axiom: typed variables are projected in parallel and each one is coerced in order to show only the *relevant* hypotheses for each future term of the parallel. This way, the terms in parallel share the same resources, that is, the same context of free typed variables, but each of the terms does it in a "customized" way. Moreover, the parallel constructor is non-idempotent and non-commutative, exactly like the intersection type operator (in its strict version, see the beginning of Section 3). These two facts imply that it is possible to make as many copies as we want of a variable in the context (thanks to non-idempotence), and in the order we prefer (thanks to non-commutativity); thus, we prepare all the necessary warps to weave the terms, each of them typed with one component of an intersection type.

Three rules allow us to generate all typed terms. The first two rules, Application and Lambda-abstraction, are performed on all the components of a parallel term in a synchronized way. Finally, via the third rule of Local Renaming, once a free typed variable is bounded by lambda-abstraction, each of the terms in parallel can do its local renaming, with type refinement according to the term's *relevant* hypotheses on that particular resource. Notice that Local Renaming corresponds to an intersection elimination, performed only at the very end of a type derivation, that is, Local Renaming can be seen as the dual of the Axiom.

Thus, we have achieved: (i) a way of representing intersection, via the relevant parallel, without ever introducing it, except at the very start of a derivation to list the resources that will be used; (ii) none of our rules require any meta-level condition on their application; (iii) each use of each resource is documented in the term via coercions.

The paper is structured as follows. Section 2 discusses logical interpretations of the intersection type construct. Sections 3 and 4 are the core of the paper, presenting the typed lambda calculus and its basic properties, respectively. Section 5 concerns the definition of a minimal version of the Intersection Type Discipline à la Curry, and Section 6 is devoted to proving that it is isomorphic to our typed lambda calculus. Section 7 formalizes the typed reduction rules. Finally, Section 8 concludes the paper by discussing earlier related approaches to typed calculi.

## 2. Intersection types from logical and computational points of view

Applications of intersection types have flourished in the past two decades, from filter-models semantics for the pure lambda calculus, [3], to intersection-based programming languages [26,23]. Different formulations of the intersection type inference system and related results are presented in [32]. A general definition of the intersection type theory and a survey on various formulations can be found in [2]. An account on more programming languages related research is presented in [36]. We now focus on the connections between intersection types and logics, which are the bridge between the classical intersection *type inference* system and the most wanted intersection *typed* system, subject of this paper.

The work [11] summarizes the studies on the relation between logics of entailments, developed for purely philosophical reasons in the early seventies, and intersection and union type theories. In particular, from the *minimal positive relevant logic* B+ derives a model of lambda calculus and combinatory logic. Recently, further progress has been made in this direction: [9] compares B+ with the semantics-based approach to subtyping introduced by Frisch, Castagna and Benzaken [12] in the definition of a type system with intersection and union. The paper [9] shows that, for the functional core of the system, such a notion of subtyping (which is defined in purely set-theoretical terms) coincides with the relevant entailment of the logic B+. But this brings us back to the result in [34], where the classical subtyping relation defined on intersection types, exploited to get a pure truth-functional inference system for combinatory logic, was proved to correspond to the implicational-conjunctive fragment of B+.

Going towards a typed intersection type system, we consider three results, one coming from the logical side, the other ones from the programming languages realm. In [4], a natural deduction reformulation of the system of [34] is presented: the resulting inference system for the lambda calculus does not have any meta-level rule and it has a strong logical counterpart, therefore is somewhat close to a typed version, but it fails to type all terms that are typable in ITD. In [23], a typed *for* construct is proposed, which gives a type variable a finite set of types to range over: also this construct turns out not to be enough to capture the intersection type inference completely. The paper [35] can be considered the seminal work that associates to intersection types the notion of a parallel composition of typed terms which codify the same untyped term, but it is still far from presenting a typed intersection type system. Much more satisfactory proposals towards a typed intersection system are [5,27,36,18], which we will compare with our approach at the end of the paper in Section 8.

## 3. The typed calculus

In our calculus, we use intersection types in their *strict* version as defined in [33] (see also [8]). In this approach, an intersection type of the shape $\tau_1 \wedge \cdots \wedge \tau_n$ ($n \geq 1$) is such that each $\tau_i$ is either a type variable or an arrow type, and each arrow type is of the shape $(\tau_1 \wedge \cdots \wedge \tau_n) \to \tau$, i.e., intersections can occur only at the left-hand side of the arrow type constructor. Furthermore, an intersection type $\tau_1 \wedge \cdots \wedge \tau_n$ is considered here as an ordered list of the types $\tau_1, \ldots, \tau_n$, thus, for instance, $\tau_1 \neq \tau_1 \wedge \tau_1$ (no idempotence) and $\tau_1 \wedge \tau_2 \neq \tau_2 \wedge \tau_1$ (no commutativity); moreover, since the intersection type operator is not a binary operator, there is no associativity (while in the standard non-strict binary formulation of the intersection type system it holds that $(\tau_1 \wedge \tau_2) \wedge \tau_3 = \tau_1 \wedge (\tau_2 \wedge \tau_3)$). This choice is motivated by our setting of explicitly typed λ-terms; more motivations for such a record-like syntax for intersection types are presented in [36]. Such a choice is enforced by the syntax of types presented below.

Let *TVar* be a countable (infinite) set of type variables, ranged over by $\alpha$ and $\beta$ (possibly with subscripts). The set $\mathscr{T}$ of types (ranged over by $\sigma$, possibly with subscripts) and its subset of *monotypes* (ranged over by $\tau$, $\delta$, and $\rho$, possibly with subscripts) are defined by the following grammar:

$$\begin{array}{rrcll} \textit{Monotypes} & \tau & ::= & \alpha \mid (\sigma \to \tau) & \\ \textit{Types} & \sigma & ::= & (\tau_1 \wedge \cdots \wedge \tau_n) & (n \geq 1). \end{array}$$

In writing types we will assume that $\wedge$ takes precedence over $\to$ and that $\to$ associates to the right. We will omit outermost brackets, unless they make the type easier to read.

**Notation 1** (*Intersection of Intersections*). Given $n$ types $\sigma_i \equiv \tau_1^i \wedge \cdots \wedge \tau_{k_i}^i$, such that $1 \leq i \leq n$, then $\sigma_1 \bigwedge \cdots \bigwedge \sigma_n$ will denote the intersection type:

$$\tau_1^1 \wedge \cdots \wedge \tau_{k_1}^1 \wedge \cdots \wedge \tau_1^n \wedge \cdots \wedge \tau_{k_n}^n.$$

**Notation 2.**

- $\tau \in (\tau_1 \wedge \cdots \wedge \tau_n)$ means that $\tau \equiv \tau_i$, for some $i$ such that $1 \leq i \leq n$ ($\tau$ is a *component of* $\tau_1 \wedge \cdots \wedge \tau_n$).
- $\sigma \sqsubseteq (\tau_1 \wedge \cdots \wedge \tau_n)$ means that $\tau_i \in \sigma$, for all $i$ such that $1 \leq i \leq n$.
- $\sigma \sqsubseteq \{\tau_1, \ldots, \tau_n\}$ means that $\tau_i \in \sigma$, for all $i$ such that $1 \leq i \leq n$.

It is important to notice that the second clause of the above notation, $\sigma_1 \sqsubseteq \sigma_2$, is different from the standard subtyping relation on intersection types that is introduced and used in the inference system in [3]. In our context, $\sigma_1 \sqsubseteq \sigma_2$ is only a notation for the property that each (monotype) component of $\sigma_2$ is also a component of $\sigma_1$. This relation is not integrated in our typing rules: for instance, $\sigma_1 \sqsubseteq \sigma_2$ and $\sigma_2 \sqsubseteq \sigma_1$ do not imply $\sigma_1 = \sigma_2$, and typed terms of type $\sigma_1$ are different from typed terms of type $\sigma_2$. We also include the third clause above, where we write $\sigma \sqsubseteq \{\tau_1, \ldots, \tau_n\}$ to denote that each monotype of the set $\{\tau_1, \ldots, \tau_n\}$ is a component of $\sigma$.

In the definition of our calculus we follow the style of the simply typed λ-calculus as presented in [15] (instead of other common formulations of typed calculi using pseudo-terms and distinct typing rules, such as in [30]).

Therefore, we start by assuming sets of variables indexed by the set of types. Let $\mathit{Var}_\sigma$ denote a denumerable set of variables for each $\sigma \in \mathscr{T}$. Any $x \in \mathit{Var}_\sigma$ is denoted as $x^\sigma$ to represent the pair $(x, \sigma)$ (*typed variable*). Notice that two typed variables $x^\sigma$ and $y^{\sigma'}$ are distinct, $x^\sigma \not\equiv y^{\sigma'}$, if $\sigma \not\equiv \sigma'$ or $x \not\equiv y$. Let *Var* denote the denumerable set of all typed variables.

We also assume a denumerable set $\mathscr{C}$ of constants $c^{\sigma \Rightarrow \tau}$ (*coercions*), that are defined as follows.

**Definition 3** (*Coercion*). For any type $\sigma$ and for any monotype $\tau$ such that $\tau \in \sigma$, the constant $c^{\sigma \Rightarrow \tau}$ denotes the coercion from $\sigma$ to $\tau$; we call $\sigma$ the *domain* and $\tau$ the *range* of $c^{\sigma \Rightarrow \tau}$. We call any coercion of the shape $c^{\tau \Rightarrow \tau}$ an *identity coercion*.

**Definition 4** (*Range Set*). Let $\Sigma$ be a finite set of coercions such that all coercions have the same domain; we define the *range set* as follows:

$$\mathrm{R}(\Sigma) \overset{def}{=} \{\tau \mid c^{\sigma \Rightarrow \tau} \in \Sigma\}.$$

Using *Var* and $\mathscr{C}$, we will define the *generating rules* (Definition 7) that generate the set of all and only the well-formed (well-typed) terms of our calculus.

Coercions are intended as type upcasts that are applied to all and only the occurrences of typed variables inside typed terms. Namely, $c^{\sigma \Rightarrow \tau}(x^\sigma)$ denotes that this occurrence of $x^\sigma$ is regarded as a term of type $\tau$ ($x^\sigma$ is called the *argument* of $c^{\sigma \Rightarrow \tau}$), by simply forgetting the additional type information contained in $\sigma$ (additional components of $\sigma$ that are different from $\tau$). We require any occurrence of a typed variable $x^\sigma$ in a term to be wrapped by a coercion that explicitly states which part of the type information $\sigma$ is relevant for this occurrence of $x^\sigma$.

A *typed context* $\Gamma$ is a finite set of distinct typed variables. Namely, if $\Gamma = x_1^{\sigma_1}, \dots, x_n^{\sigma_n}$, we will write $\Gamma = (x_1{}^{\sigma_1} : \sigma_1, \dots, x_n{}^{\sigma_n} : \sigma_n)$, to emphasize the types of the variables. The notation $\Gamma, x^\sigma : \sigma$ denotes the context $\Gamma \cup x^\sigma : \sigma$, where $x^\sigma : \sigma$ does not belong to $\Gamma$.

The set *Terms*($\Gamma$) of *typed terms* relative to $\Gamma$ is the set of well-formed terms typed starting from $\Gamma$. For readability, instead of writing $T^\sigma \in \mathit{Terms}(\Gamma)$ (i.e., $T$ is a term of type $\sigma$ relative to $\Gamma$), we will write $\Gamma \vdash T : \sigma$.

One of the main distinguishing features of our calculus is the strict parallel operator on typed terms that corresponds to the intersection operator on monotypes. In general, a parallel term of the shape $M_1{}^{\tau_1} | \cdots | M_n{}^{\tau_n}$ has type $\tau_1 \wedge \cdots \wedge \tau_n$ where each $\tau_i$ is the type corresponding to $M_i$. All typed terms $M_i{}^{\tau_i}$ are not parallel compositions of typed terms (i.e., no nested parallels are allowed), as well as all $\tau_i$ are monotypes (i.e., all $\tau_i$ are not intersections). Moreover, all $M_i{}^{\tau_i}$ share the same free variables in such a way that $M_1{}^{\tau_1} | \cdots | M_n{}^{\tau_n}$ is a typed term relative to a unique context $\Gamma$.

We exploit the following conventions for denoting terms:

- we use $M^\tau$ and $N^\tau$ (possibly with subscripts) to denote terms having monotypes;
- we use $\Pi^\sigma$ (possibly with subscripts) to denote a parallel term. Thus, for instance, $\Pi^\sigma$ denotes a parallel term of the shape

  $$M_1{}^{\tau_1} | \cdots | M_n{}^{\tau_n}, \text{ with } \sigma \equiv \tau_1 \wedge \cdots \wedge \tau_n;$$

- we use $T^\sigma$ to range over $M^\tau$, $N^\tau$ and $\Pi^\sigma$ (this is a sound notation as a type of length one is a monotype).

Now we present the *generating rules* defining formally the set of typed terms, in any given context $\Gamma$, that are all and only the well-formed expressions of our intersection typed lambda calculus. The following preliminary notation and definition are used to enhance the readability of the generating rules.

**Notation 5** (*Juxtapose Parallel Terms*). We use the following notations on parallel terms:

- If $\Pi_1{}^{\sigma_1} \equiv N_1{}^{\tau_1} | \cdots | N_k{}^{\tau_k}$ and $\Pi_2{}^{\sigma_2} \equiv M_1{}^{\rho_1} | \cdots | M_j{}^{\rho_j}$, where $\sigma_1 \equiv \tau_1 \wedge \cdots \wedge \tau_k$ and $\sigma_2 \equiv \rho_1 \wedge \cdots \wedge \rho_j$, then $\Pi_1{}^{\sigma_1} | \Pi_2{}^{\sigma_2} : \sigma_1 \bigwedge \sigma_2$ will denote the parallel term:

  $$N_1{}^{\tau_1} | \cdots | N_k{}^{\tau_k} | M_1{}^{\rho_1} | \cdots | M_j{}^{\rho_j} : \tau_1 \wedge \cdots \wedge \tau_k \wedge \rho_1 \wedge \cdots \wedge \rho_j.$$

- As a generalization, $\Pi_1{}^{\sigma_1} | \cdots | \Pi_n{}^{\sigma_n} : \sigma_1 \bigwedge \cdots \bigwedge \sigma_n$ will denote the parallel term:

  $$((\Pi_1{}^{\sigma_1} | \Pi_2{}^{\sigma_2}) | \dots) | \Pi_n{}^{\sigma_n}.$$

**Definition 6.** Given a variable $x^\sigma$ and a term $M^\tau$, we define the following sets:

- *Coerce*$(x^\sigma, M^\tau)$ is the set of all the coercions with argument $x^\sigma$ occurring in $M^\tau$;
- *Range*$(x^\sigma, M^\tau) \overset{def}{=} \mathrm{R}(\mathit{Coerce}(x^\sigma, M^\tau))$.

**Definition 7** (*Generating Rules*).

(AX) Axiom:
if $\Gamma$ contains $x^\sigma : \sigma$ then

$$\Gamma \vdash c^{\sigma \Rightarrow \tau_1}(x^\sigma) | \cdots | c^{\sigma \Rightarrow \tau_n}(x^\sigma) : \tau_1 \wedge \cdots \wedge \tau_n \quad (n \geq 1)$$

where each $\tau_i \in \sigma$, $1 \leq i \leq n$.

(SAp) Synchronized Application:
if

$$\Gamma \vdash {M_1}^{\sigma_1 \to \tau_1} | \cdots | {M_n}^{\sigma_n \to \tau_n} : (\sigma_1 \to \tau_1) \wedge \cdots \wedge (\sigma_n \to \tau_n) \quad (n \geq 1)$$

and

$${\Pi_1}^{\sigma_1} | \cdots | {\Pi_n}^{\sigma_n} : \sigma_1 \bigwedge \cdots \bigwedge \sigma_n$$

then

$$\Gamma \vdash {M_1}^{\sigma_1 \to \tau_1} {\Pi_1}^{\sigma_1} | \cdots | {M_n}^{\sigma_n \to \tau_n} {\Pi_n}^{\sigma_n} : \tau_1 \wedge \cdots \wedge \tau_n.$$

(SAb) Synchronized Abstraction:
if

$$\Gamma, x^\sigma : \sigma \vdash {M_1}^{\tau_1} | \cdots | {M_n}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n \quad (n \geq 1)$$

then

$$\Gamma \vdash \lambda x^\sigma . {M_1}^{\tau_1} | \cdots | \lambda x^\sigma . {M_n}^{\tau_n} : (\sigma \to \tau_1) \wedge \cdots \wedge (\sigma \to \tau_n).$$

(LR) Local Renaming:
if

$$\Gamma \vdash \lambda x^\sigma . {M_1}^{\tau_1} | \cdots | \lambda x^\sigma . {M_n}^{\tau_n} : (\sigma \to \tau_1) \wedge \cdots \wedge (\sigma \to \tau_n) \quad (n \geq 1)$$

then

$$\Gamma \vdash \lambda {z_1}^{\sigma_1} . {\overline{M}_1}^{\tau_1} | \cdots | \lambda {z_n}^{\sigma_n} . {\overline{M}_n}^{\tau_n} : (\sigma_1 \to \tau_1) \wedge \cdots \wedge (\sigma_n \to \tau_n)$$

where, for all $i$ such that $1 \leq i \leq n$:

- ${z_i}^{\sigma_i}$ is a fresh typed variable such that $\sigma \sqsubseteq \sigma_i \sqsubseteq \mathit{Range}(x^\sigma, {M_i}^{\tau_i})$;
- ${\overline{M}_i}^{\tau_i} = {M_i}^{\tau_i}[c^{\sigma \Rightarrow \delta}(x^\sigma) \leftarrow c^{\sigma_i \Rightarrow \delta}({z_i}^{\sigma_i})]$ for all $c^{\sigma \Rightarrow \delta}$ in $\mathit{Coerce}(x^\sigma, {M_i}^{\tau_i})$.

The notation $M^\tau[c^{\sigma \Rightarrow \delta}(x^\sigma) \leftarrow c^{\sigma' \Rightarrow \delta}(y^{\sigma'})]$ denotes the substitution of $c^{\sigma' \Rightarrow \delta}(y^{\sigma'})$ for $c^{\sigma \Rightarrow \delta}(x^\sigma)$ in $M^\tau$, if $x^\sigma$ is free in $M^\tau$ and $y^{\sigma'}$ is not free in $M^\tau$.

We will sometimes omit the superscript type from a typed term if the type is reconstructible trivially from its subterms' types, e.g., we will write $\lambda x^\sigma . M^\tau$ instead of $(\lambda x^\sigma . M^\tau)^{\sigma \to \tau}$.

The (AX) rule says that a free typed variable $x^\sigma$ (seen as a shared resource) can be used by the typed terms that are in a parallel composition. Moreover, any occurrence of that variable must be coerced by a coercion that records explicitly the component $\tau$ of $\sigma$ such that $x^\sigma$ is viewed as a term of type $\tau$ in that particular occurrence. Thus (AX) implies that all variables must always be coerced (Property 11), and, in fact, (AX) replaces the standard projection rule $\Gamma, x^\sigma : \sigma \vdash x^\sigma : \sigma$. This choice makes the calculus uniform and the rules and proofs simpler.

Then, typed terms are constructed by using the rules (SAp) and (SAb). By (SAp) simultaneous applications are performed between parallel terms. By (SAb) a simultaneous λ-abstraction acts on all the components of a parallel term, where the unique context $\Gamma$ guarantees that the same typed variable is bound in all these terms. By this rule, the resource $x^\sigma$, that was shared by ${M_1}^{\tau_1} | \cdots | {M_n}^{\tau_n}$, becomes an independent local resource for each $\lambda x^\sigma . {M_i}^{\tau_i}$.

Finally, by (LR) rule, once the variable $x^\sigma$ has been bound to a local copy for each $\lambda x^\sigma . {M_i}^{\tau_i}$, all the components of the parallel term $\lambda x^\sigma . {M_1}^{\tau_1} | \cdots | \lambda x^\sigma . {M_n}^{\tau_n}$ are free to rename that variable with a fresh name in an independent way. Notice that we require each ${z_i}^{\sigma_i}$ to be a fresh variable, thus no name clash can arise during this substitution (refresh). Furthermore, by performing this renaming of $x^\sigma$ each term $\lambda x^\sigma . {M_i}^{\tau_i}$ can discard some (all) of the typed information contained in $\sigma$ that are not used in ${M_i}^{\tau_i}$. Namely, the fresh typed variables ${z_i}^{\sigma_i}$ are such that:

- $\sigma \sqsubseteq \sigma_i$, i.e., $\sigma_i$ cannot add new monotypes with respect to the intersection type $\sigma$;
- $\sigma_i \sqsubseteq \mathit{Range}(x^\sigma, {M_i}^{\tau_i})$, i.e., $\sigma_i$ must contain at least all the monotypes that are relevant to the well-typedness of ${M_i}^{\tau_i}$.

Moreover, we notice that, by the above definition of generating rules:

- ${M_1}^{\tau_1}|\cdots|{M_n}^{\tau_n}$ in the particular case of $n = 1$ denotes the term ${M_1}^{\tau_1}$ having monotype $\tau_1$; in this case, we say that $M_1$ is not in parallel (with other terms);
- in ${M_1}^{\tau_1}|\cdots|{M_n}^{\tau_n}$ each ${M_i}^{\tau_i}$, having monotype $\tau_i$, is either a coerced variable, or an application of the shape $M^{\sigma\to\tau}\Pi^\sigma$ where $M$ has a monotype and $\Pi$ is a parallel term, or a $\lambda$-abstraction $\lambda x^\sigma.M^\tau$, where $\tau$ is a monotype.

We assume that it is always possible to rename bound variables with fresh names, by extending the standard $\alpha$-renaming on $\lambda$-calculus to our terms. Therefore, we will consider typed terms syntactically equal modulo $\alpha$-renaming of typed bound variables.

In Section 4 we will list the main structural properties of the typed terms, in particular, the Inversion Lemma.

## 3.1. Examples

The following examples show how to use the generating rules to prove that two significant typed terms belong to our calculus.

**Example 8.** This example shows a typed term of type $(\alpha\to\alpha)\wedge(\beta\to\beta)$ (representing two parallel typed versions of the identity $\lambda x.x$).

$$\vdash \lambda z^\alpha.c^{\alpha\Rightarrow\alpha}(z^\alpha)|\lambda w^\beta.c^{\beta\Rightarrow\beta}(w^\beta) : (\alpha\to\alpha)\wedge(\beta\to\beta)$$
$$\Uparrow \text{(LR)}$$
$$\vdash \lambda x^{\alpha\wedge\beta}.c^{\alpha\wedge\beta\Rightarrow\alpha}(x^{\alpha\wedge\beta})|\lambda x^{\alpha\wedge\beta}.c^{\alpha\wedge\beta\Rightarrow\beta}(x^{\alpha\wedge\beta}) : (\alpha\wedge\beta\to\alpha)\wedge(\alpha\wedge\beta\to\beta)$$
$$\Uparrow \text{(SAb)}$$
$$x^{\alpha\wedge\beta} : \alpha\wedge\beta \vdash c^{\alpha\wedge\beta\Rightarrow\alpha}(x^{\alpha\wedge\beta})|c^{\alpha\wedge\beta\Rightarrow\beta}(x^{\alpha\wedge\beta}) : \alpha\wedge\beta. \qquad \text{(AX)}$$

If we started with three (coerced) variable terms we could have generated three identities in parallel of type $(\alpha\to\alpha)\wedge(\beta\to\beta)\wedge(\alpha\wedge\beta\to\alpha)$; we show only the result of this after applying the last generating rule (LR):

$$\begin{array}{l}\vdash \lambda z^\alpha.c^{\alpha\Rightarrow\alpha}(z^\alpha)|\lambda w^\beta.c^{\beta\Rightarrow\beta}(w^\beta)|\lambda y^{\alpha\wedge\beta}.c^{\alpha\wedge\beta\Rightarrow\alpha}(y^{\alpha\wedge\beta}) :\\ \qquad (\alpha\to\alpha)\wedge(\beta\to\beta)\wedge(\alpha\wedge\beta\to\alpha).\end{array}$$

**Example 9.** This example shows a typed term of type
$(((\alpha_1\to\beta_1)\wedge\alpha_1)\to\beta_1)\wedge(((\alpha_2\to\beta_2)\wedge\alpha_2)\to\beta_2)$
representing two parallel typed versions of the auto-application $\lambda x.xx$.

$$\begin{array}{ll}\vdash & \lambda y^{(\alpha_1\to\beta_1)\wedge\alpha_1}.c^{(\alpha_1\to\beta_1)\wedge\alpha_1\Rightarrow\alpha_1\to\beta_1}(y^{(\alpha_1\to\beta_1)\wedge\alpha_1})c^{(\alpha_1\to\beta_1)\wedge\alpha_1\Rightarrow\alpha_1}(y^{(\alpha_1\to\beta_1)\wedge\alpha_1})\\ | & \lambda z^{(\alpha_2\to\beta_2)\wedge\alpha_2}.c^{(\alpha_2\to\beta_2)\wedge\alpha_2\Rightarrow\alpha_2\to\beta_2}(z^{(\alpha_2\to\beta_2)\wedge\alpha_2})c^{(\alpha_2\to\beta_2)\wedge\alpha_2\Rightarrow\alpha_2}(z^{(\alpha_2\to\beta_2)\wedge\alpha_2}) :\end{array}$$
$$(((\alpha_1\to\beta_1)\wedge\alpha_1)\to\beta_1)\wedge(((\alpha_2\to\beta_2)\wedge\alpha_2)\to\beta_2)$$
$$\Uparrow \text{(LR)}$$
$$\vdash\lambda x^\sigma.c^{\sigma\Rightarrow\alpha_1\to\beta_1}(x^\sigma)c^{\sigma\Rightarrow\alpha_1}(x^\sigma)\ |\ \lambda x^\sigma.c^{\sigma\Rightarrow\alpha_2\to\beta_2}(x^\sigma)c^{\sigma\Rightarrow\alpha_2}(x^\sigma) : (\sigma\to\beta_1)\wedge(\sigma\to\beta_2)$$
where $\sigma \equiv (\alpha_1\to\beta_1)\wedge\alpha_1\wedge(\alpha_2\to\beta_2)\wedge\alpha_2$
$$\Uparrow \text{(SAb)}$$
$$x^\sigma : \sigma \vdash c^{\sigma\Rightarrow\alpha_1\to\beta_1}(x^\sigma)c^{\sigma\Rightarrow\alpha_1}(x^\sigma)\ |\ c^{\sigma\Rightarrow\alpha_2\to\beta_2}(x^\sigma)c^{\sigma\Rightarrow\alpha_2}(x^\sigma) : \beta_1\wedge\beta_2$$
$$\Uparrow \text{(SAp)}$$
$$x^\sigma : \sigma \vdash c^{\sigma\Rightarrow\alpha_1\to\beta_1}(x^\sigma)\ |\ c^{\sigma\Rightarrow\alpha_2\to\beta_2}(x^\sigma) : (\alpha_1\to\beta_1)\wedge(\alpha_2\to\beta_2)$$
$$x^\sigma : \sigma \vdash c^{\sigma\Rightarrow\alpha_1}(x^\sigma)\ |\ c^{\sigma\Rightarrow\alpha_2}(x^\sigma) : \alpha_1\wedge\alpha_2.$$

Now we could apply, using rule (SAp), this typed term to the identity: we would need a typed parallel term made of four identity typed terms (it easy to build such terms following Example 8): $I^{\alpha_1\to\beta_1}|I^{\alpha_1}|I^{\alpha_2\to\beta_2}|I^{\alpha_2}$.

## 4. Properties

In this section we present the main structural properties of the typed terms, and, in particular, the Inversion Lemma.

**Property 10** (*Intersection Types and Parallel Terms*). *If*

$\Gamma \vdash T^\sigma : \sigma$ *and* $\sigma \equiv \tau_1 \wedge \cdots \wedge \tau_n$, *with* $n > 1$,

*then* $T^\sigma$ *is a parallel term, namely,* $T^\sigma \equiv M_1{}^{\tau_1}|\cdots|M_n{}^{\tau_n}$ *for some* $M_1{}^{\tau_1}, \ldots, M_n{}^{\tau_n}$.

**Proof.** Trivial. □

**Property 11** (*Basic Properties*). *If* $\Gamma \vdash M_1{}^{\tau_1}|\cdots|M_n{}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n$ *then:*

(1) *for any* $(x^\sigma : \sigma) \in \Gamma$ *and for any* $M_i{}^{\tau_i}$ $(1 \le i \le n)$, *each occurrence of* $x^\sigma$ *in* $M_i{}^{\tau_i}$ *is the argument of a coercion of the shape* $c^{\sigma\Rightarrow\tau}$ *where* $\tau \in \sigma$;
(2) *for any term of the shape* $N_1{}^{\rho_1}|\cdots|N_k{}^{\rho_k}$, *where for all* $i$ $(1 \le i \le k)$ *there exists* $j$ $(1 \le j \le n)$ *such that* $N_i{}^{\rho_i} \equiv M_j{}^{\tau_j}$, *the judgement*

$\Gamma \vdash N_1{}^{\rho_1}|\cdots|N_k{}^{\rho_k} : \rho_1 \wedge \cdots \wedge \rho_k$

*is derivable (i.e., if we can derive a parallel, we can also derive a permutation of it, a shorter parallel, a parallel with duplication).*

**Proof.** Both points are proved by induction on the derivation of
$\Gamma \vdash M_1{}^{\tau_1}|\cdots|M_n{}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n$. □

Intuitively, the second point of the previous property states that $\Gamma \vdash N_1{}^{\rho_1}|\cdots|N_k{}^{\rho_k} : \rho_1 \wedge \cdots \wedge \rho_k$ can be generated by applying the same sequence of generating rules that were used to obtain $\Gamma \vdash M_1{}^{\tau_1}|\cdots|M_n{}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n$, in particular:

- for permutation, we only need to re-arrange the derivations we already have;
- for a shorter parallel, we choose only the derivations we need;
- for duplication, we only need to prepare the n-uple of (coerced) copies of the sought variable by applying the projection rule (AX) accordingly.

**Lemma 12** (*Inversion Lemma*).

1. *If* $\Gamma \vdash c^{\sigma\Rightarrow\tau_1}(x^\sigma)|\cdots|c^{\sigma\Rightarrow\tau_n}(x^\sigma) : \tau_1 \wedge \cdots \wedge \tau_n$ *then* $x^\sigma : \sigma \in \Gamma$.
2. *If* $\Gamma \vdash M_1{}^{\sigma_1\to\tau_1}\Pi_1{}^{\sigma_1}|\cdots|M_n{}^{\sigma_n\to\tau_n}\Pi_n{}^{\sigma_n} : \tau_1 \wedge \cdots \wedge \tau_n$ *then*

$$\Gamma \vdash M_1{}^{\sigma_1\to\tau_1}|\cdots|M_n{}^{\sigma_n\to\tau_n} : (\sigma_1 \to \tau_1) \wedge \cdots \wedge (\sigma_n \to \tau_n)$$

*and*

$$\Gamma \vdash \Pi_1{}^{\sigma_1}|\cdots|\Pi_n{}^{\sigma_n} : \sigma_1 \bigwedge \cdots \bigwedge \sigma_n.$$

3. *If* $\Gamma \vdash \lambda x^\sigma.M_1{}^{\tau_1}|\cdots|\lambda x^\sigma.M_n{}^{\tau_n} : (\sigma \to \tau_1) \wedge \cdots \wedge (\sigma \to \tau_n)$ *then*

$$\Gamma, x^\sigma : \sigma \vdash M_1{}^{\tau_1}|\cdots|M_n{}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n.$$

4. *If* $\Gamma \vdash \lambda x_1{}^{\sigma_1}.M_1{}^{\tau_1}|\cdots|\lambda x_n{}^{\sigma_n}.M_n{}^{\tau_n} : (\sigma_1 \to \tau_1) \wedge \cdots \wedge (\sigma_n \to \tau_n)$ *such that all* $x_i{}^{\sigma_i}$ *are distinct, then there exits* $z^\sigma$ *such that*

$$\Gamma, z^\sigma : \sigma \vdash \overline{M}_1{}^{\tau_1}|\cdots|\overline{M}_n{}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n$$

*where*
- $\sigma \sqsubseteq \sigma_i \sqsubseteq Range(z^\sigma, \overline{M}_i{}^{\tau_i})$, *for all* $i$, $1 \le i \le n$;
- $\overline{M}_i{}^{\tau_i} = M_i{}^{\tau_i}[c^{\sigma_i\Rightarrow\delta}(x_i{}^{\sigma_i}) \leftarrow c^{\sigma\Rightarrow\delta}(z^\sigma)]$, *for all* $c^{\sigma_i\Rightarrow\delta}$ *in* $Coerce(x_i{}^{\sigma_i}, M_i{}^{\tau_i})$.

Table 1
Type inference rules

| | |
|---|---|
| [AX] | $\Gamma, x : \tau_1 \wedge \cdots \wedge \tau_n \Vdash x : \tau_i \qquad \forall i, 1 \le i \le n$ |
| [APP] | $\dfrac{\Gamma \Vdash M : \tau_1 \wedge \cdots \wedge \tau_n \to \tau \quad \Gamma \Vdash N : \tau_1 \quad \ldots \quad \Gamma \Vdash N : \tau_n}{\Gamma \Vdash MN : \tau}$ |
| [ABS] | $\dfrac{\Gamma, x : \sigma \Vdash M : \tau}{\Gamma \Vdash \lambda x.M : \sigma \to \tau}$ |

**Proof.** By induction on the derivation of the given judgements. We proceed by inspection of the generating rules. The only interesting cases are 3 and 4. Namely, 3 follows from (SAb) applied in the reverse order, and 4 follows from (LR) (in the reverse order) and then from 3. □

**Property 13** (*Property on Range*). *Given the typed term $\lambda x^\sigma.M^\tau$, then*

$$Range(x^\sigma, M^\tau) = Range(z^{\sigma'}, \overline{M}^\tau)$$

*where* $\overline{M}^\tau = M^\tau[c^{\sigma \Rightarrow \delta}(x^\sigma) \leftarrow c^{\sigma' \Rightarrow \delta}(z^{\sigma'})]$, *for all* $c^{\sigma \Rightarrow \delta}$ *in* $Coerce(x^\sigma, M^\tau)$, *and* $\sigma'$ *is such that either* $\sigma' \sqsubseteq \sigma$ *or* $\sigma \sqsubseteq \sigma' \sqsubseteq Range(x^\sigma, M^\tau)$.

**Proof.** Trivial. □

## 5. Type inference system

In this section we show that the intersection typed lambda calculus we presented is the typed version of the standard Intersection Type Discipline (ITD). ITD has different equivalent versions in the literature (see [32] for an overview of the various existing systems and their equivalence). In the present paper we consider its simplest formulation, originated in [8] and formalized with strict types in [33].

Type environments, denoted by $\Gamma$, are finite (possibly empty) sets of assumptions of the shape $x : \sigma$, where $x$ is an untyped variable, $\sigma$ is a type and all term variables are distinct. The environment $\Gamma, x : \sigma$ denotes the environment $\Gamma \cup \{x : \sigma\}$, where no assumption on $x$ belongs to $\Gamma$. For any λ-term $M$, type environment $\Gamma$ and type $\sigma$, $\Gamma \Vdash M : \sigma$ is a *type assertion* (read "$M$ has type $\sigma$ in $\Gamma$"). The type inference system is defined by the axioms and rules presented in Table 1.

A deduction is a tree of assertions: those at the tops of the branches are axioms, and those below are deduced from assertions immediately above by a rule (see [15] for detailed definitions on deductions). We write $\mathscr{D} : \Gamma \Vdash M : \sigma$ to denote that there is a deduction $\mathscr{D}$ whose conclusion is $\Gamma \Vdash M : \sigma$.

We emphasize the main characterizing features of our formulation of ITD. No explicit rule for introducing intersection

$$\frac{\Gamma \Vdash M : \tau_i \qquad \forall i = 1..n}{\Gamma \Vdash M : \tau_1 \wedge \cdots \wedge \tau_n}\ (\wedge I)$$

is defined in this calculus, namely, a proved assertion $\Gamma \Vdash M : \sigma$ implies that $\sigma \equiv \tau$ for some $\tau$, i.e., $\sigma$ is a monotype. Instead, such a rule is implicitly used whenever necessary, i.e., in [APP] rule. Analogously, we have no explicit intersection elimination rule such as

$$\frac{\Gamma \Vdash M : \tau_1 \wedge \cdots \wedge \tau_n}{\Gamma \Vdash M : \tau_i \qquad \forall i = 1..n}\ (\wedge E).$$

This rule is implicitly used whenever necessary, i.e., on assumptions, that is, in [AX]. As a consequence, the rules that are applied in any deduction of $\Gamma \Vdash M : \sigma$ are univocally determined by the structure of $M$.

Clearly, $(\wedge I)$ rule could be added to the inference system, without increasing its power: $\Gamma \Vdash M : \tau_1 \wedge \cdots \wedge \tau_n$ in the extended system if and only if $\Gamma \Vdash M : \tau_i$, for all $i$ such that $1 \le i \le n$, in the core system.

**Remark 14.** In the standard formulation of the intersection type inference system, intersection types are usually considered equivalent modulo reordering, idempotence and associativity. This equivalence is extended to arrow types containing intersection types. In the typed calculus, instead, this equivalence does not hold on intersection types. Thus, for the sake of simplicity, when mapping deductions to typed terms, the types of typed terms will be considered equal to the corresponding types used in the deductions under the above equivalence.

**Definition 15** (*Relevant Assumptions*). Given a deduction $\mathscr{D} : \Gamma \Vdash M : \tau$, then for all $x$ such that $x : \sigma \in \Gamma$, the set $\mathscr{R}(x, \mathscr{D})$ of the *relevant assumptions* on $x$ in $\mathscr{D}$ is defined as follows:

$$\mathscr{R}(x, \mathscr{D}) \overset{def}{=} \{\tau_i \mid \Gamma \Vdash x : \tau_i \text{ occurs in } \mathscr{D}\}.$$

**Property 16** (*Weakening on types of variables*). *For any term $M$, environment $\Gamma$, monotypes $\tau, \tau_1, \ldots, \tau_n$, if*

$$\mathscr{D} : \Gamma, x : \tau_1 \wedge \cdots \wedge \tau_n \Vdash M : \tau,$$

*then, for every $\rho_1, \ldots, \rho_k$ $(k \geq 1)$, there exists a deduction*

$$\overline{\mathscr{D}} : \Gamma, x : \tau_1 \wedge \cdots \wedge \tau_n \wedge \rho_1 \wedge \cdots \wedge \rho_k \Vdash M : \tau$$

*where $\overline{\mathscr{D}}$ is equal to $\mathscr{D}$ except that it uses the environment augmented with the $\rho_i$ $(1 \leq i \leq k)$.*

**Proof.** Trivial, by induction on the structure of $M$. □

**Property 17** (*Relevant Environments*). *For any term $M$, environment $\Gamma$, type $\sigma$ and monotype $\tau$, if*

$$\mathscr{D} : \Gamma, x : \sigma \Vdash M : \tau,$$

*then, for any $\sigma'$ such that $\sigma \sqsubseteq \sigma' \sqsubseteq \mathscr{R}(x, \mathscr{D})$, there exists a deduction*

$$\mathscr{D}' : \Gamma, x : \sigma' \Vdash M : \tau$$

*where $\mathscr{D}'$ is equal to $\mathscr{D}$ except that it uses the environment with the assertion $x : \sigma'$.*

**Proof.** Trivial, by induction on the structure of $M$. □

In the following section we prove the equivalence between ITD and the intersection typed lambda calculus presented in the previous sections.

## 6. Deductions and typed terms

First of all we consider $\Lambda_|$, the λ-calculus $\Lambda$ extended with terms in parallel. Terms of $\Lambda_|$ are defined as follows:

$$M ::= x \mid (\lambda x.M) \mid (MM) \mid (M| \cdots |M).$$

Arbitrary $\Lambda_|$ terms are denoted by $M$ and $N$ with or without subscripts. Clearly the set $\Lambda$ of λ-terms is a proper subset of $\Lambda_|$ (erase the last production in the grammar above).

We use the standard convention on λ-terms for omitting parentheses when writing $\Lambda_|$ terms. Concerning parallel terms $M_1| \cdots |M_n$ $(n > 1)$, they will always be enclosed by parentheses to avoid ambiguities when they are subterms of other terms; we only omit outermost parentheses.

As far as pure λ-terms are concerned, we consider syntactic identity modulo $\alpha$-renaming, denoted with $\equiv_\alpha$. We extend this syntactic equivalence $\equiv_\alpha$ to parallel terms in a trivial way: $\Lambda_|$ are considered identical modulo $\alpha$-renaming of bound variables. Thus, we can use the usual notation $M[x \leftarrow N]$ to denote the result of substituting $N$ for each free occurrence of $x$ in $N$ and making any renaming of bound variables needed to avoid clashes.

**Definition 18** (*Parallel Collapse*). The function $()^* : \Lambda_| \to \Lambda$ that collapses parallel terms is defined as follows:

- $(x)^* = x$;
- $(MN)^* = (M)^*(N)^*$;
- $(\lambda x.M)^* = \lambda x.(M)^*$;
- $(M_1| \cdots |M_n)^* = (M_1)^*$.

**Definition 19** (*Equivalence* $\sim$). The equivalence relation $\sim$ on $\Lambda_|$ terms is defined as follows:

- if $M$ and $N$ are pure $\lambda$-terms ($\in \Lambda$) then $M \sim N$ if and only if $M \equiv_\alpha N$;
- otherwise:
  . $\lambda x.M \sim \lambda y.N$ if and only if $M \sim (N[y \leftarrow x])$;
  . $MN \sim M'N'$ if and only if $M \sim M'$ and $N \sim N'$;
  . $M_1|\cdots|M_n \sim N_1|\cdots|N_k$ $(n, k \geq 1)$ if and only if $M_1 \sim \cdots \sim M_n \sim N_1 \sim \cdots \sim N_k$.

The type erasure function $E : \mathit{Terms} \to \Lambda_|$, given a typed term $M^\tau$, returns the untyped $\Lambda_|$ term obtained by removing all the coercions and all the type annotations from $M^\tau$.

**Definition 20** (*Type Erasure Function* $E$). The type erasure function $E : \mathit{Terms} \to \Lambda_|$ is defined as follows:

- $E(c^{\sigma \Rightarrow \tau}(x^\sigma)) = x$;
- $E(\lambda x^\sigma.M^\tau) = \lambda x.E(M^\tau)$;
- $E({M_1}^{\tau_1}|\cdots|{M_n}^{\tau_n}) = E(M_1^{\tau_1})|\cdots|E(M_n^{\tau_n})$.

The following lemma shows that, by applying the type erasure $E$ to any typed term $M^\tau$, we obtain a $\Lambda_|$ term such that all components of a parallel composition are equivalent.

**Lemma 21** (*Structural Equivalence of Terms in Parallel*). *If*

$$\Gamma \vdash {M_1}^{\tau_1}|\cdots|{M_n}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n$$

*then*

$$E({M_1}^{\tau_1}) \sim E({M_2}^{\tau_2}) \sim \cdots \sim E({M_n}^{\tau_n}).$$

**Proof.** By structural induction on ${M_1}^{\tau_1}|\cdots|{M_n}^{\tau_n}$ using generating rules. The only interesting case is (LR):

$$\Gamma \vdash \lambda {x_1}^{\sigma_1}.{M_1}^{\tau_1}|\cdots|\lambda {x_n}^{\sigma_n}.{M_n}^{\tau_n} : (\sigma_1 \to \tau_1) \wedge \cdots \wedge (\sigma_n \to \tau_n).$$

By using the Inversion Lemma (Lemma 12) we obtain

$$\Gamma, z^\sigma : \sigma \vdash \overline{M_1}^{\tau_1}|\cdots|\overline{M_n}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n.$$

By induction hypothesis, all the $E(\overline{M_i}^{\tau_i})$ are equivalent. Thus also $E(\lambda z^\sigma.\overline{M_i}^{\tau_i})$ are all equivalent. By the definition of $\overline{M}_i$ (see Lemma 12) and by the definition of $E$, also $E(\lambda z^\sigma.\overline{M_i}^{\tau_i}) \sim E(\lambda {x_i}^{\sigma_i}.{M_i}^{\tau_i})$ for all $i$ such that $1 \leq i \leq n$, thus all the $E(\lambda {x_i}^{\sigma_i}.{M_i}^{\tau_i})$ are equivalent. □

Thus, since all typed terms in parallel represent the same underlying untyped $\lambda$-term (modulo $\alpha$-renaming), by Lemma 21, it makes sense to remove also the | term constructor, by identifying the components that are in parallel with one (any) of them. For this aim, we extend the erasure $E$ to the type erasure function $\mathscr{E} : \mathit{Terms} \to \Lambda$.

**Definition 22** (*Type Erasure Function* $\mathscr{E}$). The type erasure function $\mathscr{E} : \mathit{Terms} \to \Lambda$ is defined as follows:

$$\mathscr{E}(M^\tau) \overset{def}{=} (E(M^\tau))^*.$$

**Notation 23.**

- If $\Gamma$ is a type environment, then $\Gamma_t$ represents the corresponding typed context

$$\Gamma_t = \{x^\sigma : \sigma \mid x : \sigma \in \Gamma\}.$$

- Conversely, if $\Gamma$ is a typed context, then $\Gamma_u$ represents the corresponding (untyped) type environment

$$\Gamma_u = \{x : \sigma \mid x^\sigma : \sigma \in \Gamma\}.$$

- For the sake of simplicity, we assume that all typed variables have different variable names; thus we avoid cases where we have two distinct typed variables $x^\sigma$ and $x^{\sigma'}$ with $\sigma \neq \sigma'$ that should be differently renamed when translating them into (untyped) term variables.

**Theorem 24** (*From Deductions to Typed Terms*). *Given a set of deductions*

$$\mathscr{D}_1 : \Gamma \Vdash M : \tau_1$$
$$\vdots$$
$$\mathscr{D}_n : \Gamma \Vdash M : \tau_n \qquad (n \geq 1)$$

*there exists a typed term*

$$\Gamma_t \vdash {M_1}^{\tau_1} | \cdots | {M_n}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n$$

*such that*

1. $M \equiv_\alpha \mathscr{E}({M_1}^{\tau_1}) \equiv_\alpha \mathscr{E}({M_2}^{\tau_2}) \equiv_\alpha \ldots \equiv_\alpha \mathscr{E}({M_n}^{\tau_n})$;
2. *Range*$(x^\sigma, {M_i}^{\tau_i}) = \mathscr{R}(x, \mathscr{D}_i)$ *for each* $x^\sigma$ *such that* $x^\sigma : \sigma \in \Gamma_t$ *and for all* $i$ *such that* $1 \leq i \leq n$.

**Proof.** By structural induction on $M$, which corresponds to an induction on the sum of the depths of the deductions $\mathscr{D}_i$ (since deduction rules are syntax driven).

- $M \equiv x$.
  We have the deductions $\mathscr{D}_i : \Gamma, x : \sigma \Vdash x : \tau_i$ with $1 \leq i \leq n$. Their existence implies that $\tau_i \in \sigma$, for all $i$ such that $1 \leq i \leq n$. Thus the typed term is

$$\Gamma_t, x^\sigma : \sigma \vdash c^{\sigma \Rightarrow \tau_1}(x^\sigma) | \cdots | c^{\sigma \Rightarrow \tau_n}(x^\sigma) : \tau_1 \wedge \cdots \wedge \tau_n.$$

- $M \equiv MN$.
  In any deduction $\mathscr{D}_i : \Gamma \Vdash MN : \tau_i$ the last applied rule is [APP], i.e.,

$$\frac{\begin{array}{c}\Gamma \Vdash M : \rho^i_1 \wedge \cdots \wedge \rho^i_{k_i} \to \tau_i \\ \Gamma \Vdash N : \rho^i_1 \qquad \ldots \qquad \Gamma \Vdash N : \rho^i_{k_i}\end{array}}{\Gamma \Vdash MN : \tau_i}\,[\text{APP}].$$

  Now, let $\sigma_i \equiv \rho^i_1 \wedge \cdots \wedge \rho^i_{k_i}$. By induction hypothesis we have

$$\Gamma_t \vdash {M_1}^{\sigma_1 \to \tau_1} | \cdots | {M_n}^{\sigma_n \to \tau_n} : (\sigma_1 \to \tau_1) \wedge \cdots \wedge (\sigma_n \to \tau_n),$$
$$\Gamma_t \vdash {\Pi_1}^{\sigma_1} | \cdots | {\Pi_n}^{\sigma_n} : \sigma_1 \bigwedge \cdots \bigwedge \sigma_n \quad \text{where } {\Pi_i}^{\sigma_i} \equiv {N_{i\,1}}^{\rho^i_1} | \cdots | {N_{i\,k_i}}^{\rho^i_{k_i}}$$

  for which the theorem assertions hold. Thus, we can apply (SAp):

$$\Gamma_t \vdash {M_1}^{\sigma_1 \to \tau_1} {\Pi_1}^{\sigma_1} | \cdots | {M_n}^{\sigma_n \to \tau_n} {\Pi_n}^{\sigma_n} : \tau_1 \wedge \cdots \wedge \tau_n.$$

  Points 1 and 2 follow from the induction hypothesis (in particular, point 2 holds on each ${M_i}^{\sigma_i \to \tau_i} {\Pi_i}^{\sigma_i}$, by induction hypothesis).
- $M \equiv \lambda x.M$.
  Consider the deductions $\mathscr{D}_i : \Gamma \Vdash \lambda x.M : \sigma_i \to \tau_i$, $i = 1..n$. The last applied rule in each deduction is

$$\frac{\Gamma, x : \sigma_i \Vdash M : \tau_i}{\Gamma \Vdash \lambda x.M : \sigma_i \to \tau_i}\,[\text{ABS}].$$

  By Property 16, we have that for each $\mathscr{D}_i$ there exists a $\overline{\mathscr{D}_i}$ such that the last applied rule in $\overline{\mathscr{D}_i}$ is

$$\frac{\Gamma, x : \sigma_1 \bigwedge \cdots \bigwedge \sigma_n \Vdash M : \tau_i}{\Gamma \Vdash \lambda x.M : \sigma_i \to \tau_i}\,[\text{ABS}]$$

  where $(*)$ $\mathscr{R}(x, \overline{\mathscr{D}_i}) \equiv \mathscr{R}(x, \mathscr{D}_i)$. Let $\sigma = \sigma_1 \bigwedge \cdots \bigwedge \sigma_n$.
  By induction hypothesis, we have

$$(**) \qquad \Gamma_t, x^\sigma : \sigma \vdash {M_1}^{\tau_1} | \cdots | {M_n}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n$$

where $Range(x^\sigma, M_i{}^{\tau_i}) \equiv \mathscr{R}(x, \overline{\mathscr{D}}_i)$ and, by $(*)$ also $Range(x^\sigma, M_i{}^{\tau_i}) \equiv \mathscr{R}(x, \mathscr{D}_i)$. We then use (SAb) to obtain

$$\Gamma_t \vdash \lambda x^\sigma.M_1{}^{\tau_1} | \cdots | \lambda x^\sigma.M_n{}^{\tau_n} : (\sigma \to \tau_1) \wedge \cdots \wedge (\sigma \to \tau_n).$$

Since $Range(x^\sigma, M_i{}^{\tau_i}) \equiv \mathscr{R}(x, \overline{\mathscr{D}}_i)$, we can apply (LR) to have

$$\Gamma_t \vdash \lambda z_1{}^{\sigma_1}.\overline{M}_1{}^{\tau_1} | \cdots | \lambda z_n{}^{\sigma_n}.\overline{M}_n{}^{\tau_n} : (\sigma_1 \to \tau_1) \wedge \cdots \wedge (\sigma_n \to \tau_n)$$

where $\mathscr{E}(\lambda z_i{}^{\sigma_i}.\overline{M}_i{}^{\tau_i}) \equiv_\alpha \mathscr{E}(\lambda x^\sigma.M_i{}^{\tau_i})$.

Finally, (2) follows from the property (2) on $\Gamma_t$ by induction hypothesis $(**)$ and by Property 13. □

**Theorem 25** (*From Typed Terms to Deductions*). *Given the typed term*

$$\Gamma \vdash M_1{}^{\tau_1} | \cdots | M_n{}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n \qquad (n \geq 1)$$

*there exist a term $M$ and $n$ deductions $\mathscr{D}_1, \ldots, \mathscr{D}_n$*

$$\begin{aligned} &\mathscr{D}_1 : \Gamma_u \Vdash M : \tau_1 \\ &\vdots \\ &\mathscr{D}_n : \Gamma_u \Vdash M : \tau_n \end{aligned}$$

*such that*

1. $M \equiv_\alpha \mathscr{E}(M_1{}^{\tau_1}) \equiv_\alpha \mathscr{E}(M_2{}^{\tau_2}) \equiv_\alpha \ldots \equiv_\alpha \mathscr{E}(M_n{}^{\tau_n})$;
2. *$Range(x^\sigma, M_i{}^{\tau_i}) = \mathscr{R}(x, \mathscr{D}_i)$ for each $x$ such that $x : \sigma \in \Gamma_u$ and for all $i$ such that $1 \leq i \leq n$.*

**Proof.** By induction on the generating rules.

- The case of (AX) is trivial.
- Case (SAp). We have

$$\Gamma \vdash M_1{}^{\sigma_1 \to \tau_1} \Pi_1{}^{\sigma_1} | \cdots | M_n{}^{\sigma_n \to \tau_n} \Pi_n{}^{\sigma_n} : \tau_1 \wedge \cdots \wedge \tau_n$$

where $\sigma_i \equiv \rho^i_1 \wedge \cdots \wedge \rho^i_{k_i}$ and $\Pi_i^{\sigma_i} \equiv N_{i\,1}{}^{\rho^i_1} | \cdots | N_{i\,k_i}{}^{\rho^i_{k_i}}$.

By Inversion Lemma

$$\Gamma \vdash M_1{}^{\sigma_1 \to \tau_1} | \cdots | M_n{}^{\sigma_n \to \tau_n} : (\sigma_1 \to \tau_1) \wedge \cdots \wedge (\sigma_n \to \tau_n),$$
$$\Gamma \vdash \Pi_1{}^{\sigma_1} | \cdots | \Pi_n{}^{\sigma_n} : \sigma_1 \bigwedge \cdots \bigwedge \sigma_n.$$

By induction hypothesis there exist the deductions:

$$\mathscr{D}_i : \Gamma_u \Vdash M : \sigma_i \to \tau_i \qquad i = 1..n$$
$$\mathscr{D}_j : \Gamma_u \Vdash N : \rho^i_j \qquad j = 1..k_i$$

where the assertions 1 and 2 hold. Now we can apply [APP] to obtain the desired $n$ deductions:

$$\frac{\begin{array}{c}\Gamma_u \Vdash M : \rho^i_1 \wedge \cdots \wedge \rho^i_{k_i} \to \tau_i \\ \Gamma_u \Vdash N : \rho^i_1 \qquad \ldots \qquad \Gamma_u \Vdash N : \rho^i_{k_i}\end{array}}{\Gamma_u \Vdash MN : \tau_i}\,[\mathrm{APP}]$$

where 1 and 2 hold by the induction hypothesis.
- Case (SAb). Similar to the previous case: use the inversion lemma, then the induction hypothesis and finally rule [ABS].
- Case (LR). We have

$$\Gamma \vdash \lambda x_1{}^{\sigma_1}.M_1{}^{\tau_1} | \cdots | \lambda x_n{}^{\sigma_n}.M_n{}^{\tau_n} : (\sigma_1 \to \tau_1) \wedge \cdots \wedge (\sigma_n \to \tau_n)$$

and, by the Inversion Lemma,

$$\Gamma, x^\sigma : \sigma \vdash \overline{M}_1{}^{\tau_1} | \cdots | \overline{M}_n{}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n$$

where $\sigma \sqsubseteq \sigma_i \sqsubseteq Range(x^\sigma, \overline{M_i}^{\tau_i})$, for all $i$ such that $1 \le i \le n$ (and $\overline{M_i}$ are defined as in Lemma 12). By induction hypothesis, we have $n$ deductions $\mathscr{D}_1, \dots, \mathscr{D}_n$ such that:

$$\mathscr{D}_i : \Gamma_u, x : \sigma \Vdash M : \tau_i \quad (1 \le i \le n)$$

where the assertions 1 and 2 hold. Moreover, for each $\mathscr{D}_i$, $Range(x^\sigma, \overline{M_i}^{\tau_i}) = Range(x^\sigma, M_i{}^{\tau_i})$ (by Property 13) and $Range(x^\sigma, M_i{}^{\tau_i}) = \mathscr{R}(x, \mathscr{D}_i)$ (by (2) of the induction hypothesis), then we obtain $\sigma \sqsubseteq \sigma_i \sqsubseteq \mathscr{R}(x, \mathscr{D}_i)$.

Thus, using Property 17 on all$\mathscr{D}_i$ we obtain the deductions $\mathscr{D}'_i(1 \le i \le n)$:

$$\mathscr{D}'_i : \Gamma_u, x : \sigma_i \Vdash M : \tau_i.$$

The thesis then follows by adding an application of [ABS] to each $\mathscr{D}'_i$; it is easy to verify that $\Gamma_u$ still satisfies (2) in each $\mathscr{D}'_i$. □

**Corollary 26.** *If $\Gamma \Vdash M : \tau_1 \wedge \cdots \wedge \tau_n$ in the inference system extended with $(\wedge I)$ rule, then $\Gamma_t \vdash M_1{}^{\tau_1} | \cdots | M_n{}^{\tau_n} : \tau_1 \wedge \cdots \wedge \tau_n$, where $M \equiv_\alpha \mathscr{E}(M_i{}^{\tau_i})$ for all $i$ such that $1 \le i \le n$.*

**Proof.** By Theorems 24 and 25. □

**Example 27.** Let us consider Example 9 of Section 3.1, where we proved that the following is a typed term:

$$\begin{array}{ll} \vdash & \lambda y^{(\alpha_1 \to \beta_1) \wedge \alpha_1}.c^{(\alpha_1 \to \beta_1) \wedge \alpha_1 \Rightarrow \alpha_1 \to \beta_1}(y^{(\alpha_1 \to \beta_1) \wedge \alpha_1})c^{(\alpha_1 \to \beta_1) \wedge \alpha_1 \Rightarrow \alpha_1}(y^{(\alpha_1 \to \beta_1) \wedge \alpha_1}) \\ | & \lambda z^{(\alpha_2 \to \beta_2) \wedge \alpha_2}.c^{(\alpha_2 \to \beta_2) \wedge \alpha_2 \Rightarrow \alpha_2 \to \beta_2}(z^{(\alpha_2 \to \beta_2) \wedge \alpha_2})c^{(\alpha_2 \to \beta_2) \wedge \alpha_2 \Rightarrow \alpha_2}(z^{(\alpha_2 \to \beta_2) \wedge \alpha_2}) : \\ & \qquad (((\alpha_1 \to \beta_1) \wedge \alpha_1) \to \beta_1) \wedge (((\alpha_2 \to \beta_2) \wedge \alpha_2) \to \beta_2) \end{array} .$$

The above term encodes the following pair of deductions $\mathscr{D}_1$ and $\mathscr{D}_2$ where:
$\mathscr{D}_1 : \Vdash \lambda x.xx : ((\alpha_1 \to \beta_1) \wedge \alpha_1) \to \beta_1$
$\mathscr{D}_2 : \Vdash \lambda x.xx : ((\alpha_2 \to \beta_2) \wedge \alpha_2) \to \beta_2$.
which are constructed in parallel by applying the same rule at each step ($\sigma_1 \equiv (\alpha_1 \to \beta_1) \wedge \alpha_1$ and $\sigma_2 \equiv (\alpha_2 \to \beta_2) \wedge \alpha_2$):

$$\dfrac{\dfrac{x : \sigma_1 \Vdash x : (\alpha_1 \to \beta_1) \quad x : \sigma_1 \Vdash x : \alpha_1}{x : \sigma_1 \Vdash xx : \beta_1}[\text{APP}]}{\Vdash \lambda x.xx : \sigma_1 \to \beta_1}[\text{ABS}] \qquad \dfrac{\dfrac{x : \sigma_2 \Vdash x : (\alpha_2 \to \beta_2) \quad x : \sigma_2 \Vdash x : \alpha_2}{x : \sigma_2 \Vdash xx : \beta_2}[\text{APP}]}{\Vdash \lambda x.xx : \sigma_2 \to \beta_2}[\text{ABS}] .$$

Conversely, the encoding of $\mathscr{D}_1$ and $\mathscr{D}_2$ in parallel, defined by the typed term above, can be obtained as follows. We first start from the context $\Gamma = \sigma_1 \bigwedge \sigma_2$. Then, the encoding is constructed bottom-up, by using the generating rules, as in Example 9. Namely, [APP] in $\mathscr{D}_1$ and $\mathscr{D}_2$ corresponds to (SAp) on the coerced variables in parallel, [ABS] corresponds to (SAb) and a final (LR) recovers the fact that different assumptions on $x$ are used in $\mathscr{D}_1$ and $\mathscr{D}_2$, respectively.

## 7. Reducing typed terms

Every reduction on a component of a parallel term must correspond to the same reduction on the other components. Thus, we give a reduction strategy that acts in a synchronized way on all the components of a parallel term. Notice that the choice of the strategy is not relevant: the reduction of typed terms will correspond to a reduction on the corresponding λ-terms and every λ-term that has a type in the deduction system is strongly normalizable.

Our choice of operational semantics is the usual one-step *eager* or *call-by-value* reduction for λ-calculus (see, e.g., [22, p.93]). We extend this reduction strategy to the Intersection Typed Lambda Calculus here presented: we only need to perform each reduction step on all the terms that are in parallel, simultaneously. Notice that this reduction is not intended to produce normal forms or to fully reduce open terms that may have free variables.

**Definition 28** (*Values*). Typed values, ranged over by $v$ and $u$, are defined as follows:

- $c^{\sigma \Rightarrow \tau_1}(x^\sigma) | \cdots | c^{\sigma \Rightarrow \tau_n}(x^\sigma)$ is a value;
- $\lambda x_1{}^{\sigma_1}.M_1{}^{\tau_1} | \cdots | \lambda x_n{}^{\sigma_n}.M_n{}^{\tau_n}$ is a value.

Table 2
Reduction rules for typed terms

$$\frac{M_1{}^{\sigma_1\to\tau_1}|\cdots|M_n{}^{\sigma_n\to\tau_n} \longrightarrow N_1{}^{\sigma_1\to\tau_1}|\cdots|N_n{}^{\sigma_n\to\tau_n}}{M_1{}^{\sigma_1\to\tau_1}\Pi_1{}^{\sigma_1}|\cdots|M_n{}^{\sigma_n\to\tau_n}\Pi_n{}^{\sigma_n} \longrightarrow N_1{}^{\sigma_1\to\tau_1}\Pi_1{}^{\sigma_1}|\cdots|N_n{}^{\sigma_n\to\tau_n}\Pi_n{}^{\sigma_n}}\ (\mathrm{R1})$$

$$\frac{\Pi_1{}^{\sigma_1}|\cdots|\Pi_n{}^{\sigma_n} \longrightarrow \Pi'_1{}^{\sigma_1}|\cdots|\Pi'_n{}^{\sigma_n}}{v_1{}^{\sigma_1\to\tau_1}\Pi_1{}^{\sigma_1}|\cdots|v_n{}^{\sigma_n\to\tau_n}\Pi_n{}^{\sigma_n} \longrightarrow v_1{}^{\sigma_1\to\tau_1}\Pi'_1{}^{\sigma_1}|\cdots|v_n{}^{\sigma_n\to\tau_n}\Pi'_n{}^{\sigma_n}}\ (\mathrm{R2})$$

$$\frac{}{(\lambda x_1{}^{\sigma_1}.M_1{}^{\tau_1})v_1{}^{\sigma_1}|\cdots|(\lambda x_n{}^{\sigma_n}.M_n{}^{\tau_n})v_n{}^{\sigma_n} \longrightarrow N_1{}^{\tau_1}|\cdots|N_n{}^{\tau_n}}\ (\mathrm{App})$$
where $N_i{}^{\tau_i} = M_i{}^{\tau_i}[c^{\sigma_i\Rightarrow\delta}(x_i{}^{\sigma_i}) \leftarrow \langle v_i{}^{\sigma_i}\rangle_\delta]$ for all $c^{\sigma_i\Rightarrow\delta}$ in $\mathit{Coerce}(x_i{}^{\sigma_i}, M_i{}^{\tau_i})$

**Notation 29** (*Juxtapose Parallel Values*). We extend Notation 1 to parallel values:

$$\pi_1{}^{\sigma_1}|\cdots|\pi_n{}^{\sigma_n} : \sigma_1 \bigwedge\cdots\bigwedge \sigma_n$$

where $\pi_i$ are values in parallel $v_{i\,1}{}^{\tau_1^i}|\cdots|v_{i\,k_i}{}^{\tau_{k_i}^i}$ and $\sigma_i \equiv \tau_1^i \wedge\cdots\wedge \tau_{k_i}^i$.

**Definition 30** (*Projection on Typed Parallel Values*). If

$$v \equiv v_1{}^{\tau_1}|\cdots|v_n{}^{\tau_n} : \tau_1\wedge\cdots\wedge\tau_n$$

is a typed parallel value and $\tau \in (\tau_1\wedge\cdots\wedge\tau_n)$, we define $\langle v_1{}^{\tau_1}|\cdots|v_n{}^{\tau_n}\rangle_\tau$ as follows:

$$\langle v_1{}^{\tau_1}|\cdots|v_n{}^{\tau_n}\rangle_\tau \overset{def}{=} \begin{cases} v_1{}^{\tau_1} & \text{if } \tau\equiv\tau_1 \\ \langle v_2{}^{\tau_2}|\cdots|v_n{}^{\tau_n}\rangle_\tau & \text{otherwise.}\end{cases}$$

The reduction rules are defined in Table 2. The most important step in the reduction procedure consists in applying the rule (App) that performs simultaneously the standard contraction of the typed redexes that are in a parallel composition, i.e.,

$$(\lambda x_1{}^{\sigma_1}.M_1{}^{\tau_1})v_1{}^{\sigma_1}|\cdots|(\lambda x_n{}^{\sigma_n}.M_n{}^{\tau_n})v_n{}^{\sigma_n}.$$

This means that, for each $i$, all free occurrences of $x_i{}^{\sigma_i}$ in $M_i{}^{\tau_i}$ will be replaced by $v_i{}^{\sigma_i}$. By the definition of typed terms, any occurrence of $x_i{}^{\sigma_i}$ in $M_i{}^{\tau_i}$ is wrapped by a coercion, say $c^{\sigma_i\Rightarrow\tau}$, such that $c^{\sigma_i\Rightarrow\tau}(x_i{}^{\sigma_i})$ is a subterm of type $\tau$ and $\tau\in\sigma_i$; moreover, if $\sigma_i$ is an intersection of monotypes, then $v_i{}^{\sigma_i}$ is a parallel term having a component (which is not a parallel) of type $\tau$. Therefore, the contraction of the redex $(\lambda x_i{}^{\sigma_i}.M_i{}^{\tau_i})v_i{}^{\sigma_i}$ will consist in replacing the subterm $c^{\sigma_i\Rightarrow\tau}(x_i{}^{\sigma_i})$ by a component of $v_i{}^{\sigma_i}$ having type $\tau$, i.e., $\langle v_i{}^{\sigma_i}\rangle_\tau$, for all free occurrences of $x_i{}^{\sigma_i}$ in $M_i{}^{\tau_i}$. Moreover, since typed terms are considered syntactically equal modulo typed $\alpha$-renaming of bound variables, then we can assume that no bound-variable clashes will arise during this substitution. Notice that we substitute directly to the (coerced) variable the projection of the argument. Thus, during the reduction, the property that each variable is the argument of a coercion still holds (Property 11), by the definition of values (Definition 28) and by the definition of projection (Definition 30).

**Lemma 31** (*Typed Substitution Lemma*). *If*

$$\Gamma, x^\sigma : \sigma \vdash M_1{}^{\tau_1}|\cdots|M_n{}^{\tau_n} : \tau_1\wedge\cdots\wedge\tau_n$$

*and $v$ is a typed value such that*

$$\Gamma \vdash v \equiv \pi_1{}^{\sigma_1}|\cdots|\pi_n{}^{\sigma_n} : \sigma_1\bigwedge\cdots\bigwedge\sigma_n \textit{ with } \sigma_i \sqsubseteq \mathrm{R}(\mathit{Coerce}(x^\sigma, M_i{}^{\tau_i}))$$

*then*

$$\Gamma \vdash N_1{}^{\tau_1}|\cdots|N_n{}^{\tau_n} : \tau_1\wedge\cdots\wedge\tau_n$$
*where* $N_i{}^{\tau_i} = M_i{}^{\tau_i}[c^{\sigma\Rightarrow\delta}(x^\sigma) \leftarrow \langle\pi_i{}^{\sigma_i}\rangle_\delta]$, *for all* $c^{\sigma\Rightarrow\delta}$ *in* $\mathit{Coerce}(x^\sigma, M_i{}^{\tau_i})$.

**Proof.** By structural induction on ${M_1}^{\tau_1}|\cdots|{M_n}^{\tau_n}$.

Base ${M_1}^{\tau_1}|\cdots|{M_n}^{\tau_n}$ are coercions on a typed variable. The only non-void case is ${M_1}^{\tau_1}|\cdots|{M_n}^{\tau_n} \equiv c^{\sigma\Rightarrow\tau_1}(x^\sigma)|\cdots|c^{\sigma\Rightarrow\tau_n}(x^\sigma)$, where $\tau_i \in \sigma$ for all $i$ such that $1 \le i \le n$.

Since $\mathrm{R}(\mathit{Coerce}(x^\sigma, {M_i}^{\tau_i})) = \{\tau_i\}$ and $\sigma_i \sqsubseteq \mathrm{R}(\mathit{Coerce}(x^\sigma, {M_i}^{\tau_i}))$, then $\tau_i \in \sigma_i$ and we have that $\langle{\pi_i}^{\sigma_i}\rangle_{\tau_i}$ is defined and so is ${\overline{M}_1}^{\tau_1}|\cdots|{\overline{M}_n}^{\tau_n} = \langle{\pi_1}^{\sigma_1}\rangle_{\tau_1}|\cdots|\langle{\pi_n}^{\sigma_n}\rangle_{\tau_n}$. Then, by applying Property 11 on the assumption $\Gamma \vdash {\pi_1}^{\sigma_1}|\cdots|{\pi_n}^{\sigma_n} : \sigma_1 \bigwedge\cdots\bigwedge \sigma_n$ we have $\Gamma \vdash \langle{\pi_1}^{\sigma_1}\rangle_{\tau_1}|\cdots|\langle{\pi_n}^{\sigma_n}\rangle_{\tau_n} : \tau_1 \wedge\cdots\wedge \tau_n$.

Induction step

- $\Gamma, x^\sigma : \sigma \vdash \lambda y^{\sigma'}.N_1^{\tau_1'}|\cdots|\lambda y^{\sigma'}.N_n^{\tau_n'} : \tau_1 \wedge\cdots\wedge \tau_n$, where $x^\sigma \not\equiv y^{\sigma'}$ and $\tau_i \equiv \sigma' \to \tau_i'$.

  By Inversion Lemma 12 we have

  $$\Gamma, x^\sigma : \sigma, y^{\sigma'} : \sigma' \vdash N_1^{\tau_1'}|\cdots|N_n^{\tau_n'} : \tau_1' \wedge\cdots\wedge \tau_n'.$$

  Then the result follows by applying the rule (SAb) to the induction hypothesis.
- $\Gamma, x^\sigma : \sigma \vdash \lambda {x_1}^{\sigma_1}.N_1^{\tau_1'}|\cdots|\lambda {x_n}^{\sigma_n}.N_n^{\tau_n'} : \tau_1 \wedge\cdots\wedge \tau_n$ where ${x_i}^{\sigma_i} \not\equiv x^\sigma$ and $\tau_i \equiv \sigma_i \to \tau_i'$.

  The proof is similar to the previous case: use Inversion Lemma, induction hypothesis, (SAb) generating rule and finally (LR) to reconstruct the term.
- $\Gamma, x^\sigma : \sigma \vdash {N_1}^{\sigma_1\to\tau_1}{\Pi_1}^{\sigma_1}|\cdots|{N_n}^{\sigma_n\to\tau_n}{\Pi_n}^{\sigma_n} : \tau_1 \wedge\cdots\wedge \tau_n$.

  Use Inversion Lemma, induction hypothesis and (SAp) generating rule. □

**Theorem 32** (*Subject Reduction*). *If $\Gamma \vdash M : \sigma$ and $M \longrightarrow M'$, then $\Gamma \vdash M' : \sigma$.*

**Proof.** We proceed by induction on the (depth of) the reduction $M \longrightarrow M'$, with case analysis on the last applied rule.

Base $(\lambda {x_1}^{\sigma_1}.{M_1}^{\tau_1}){\pi_1}^{\sigma_1}|\cdots|(\lambda {x_n}^{\sigma_n}.{M_n}^{\tau_n}){\pi_n}^{\sigma_n} \longrightarrow {N_1}^{\tau_1}|\cdots|{N_n}^{\tau_n}$, where ${N_i}^{\tau_i} = {M_i}^{\tau_i}[c^{\sigma_i\Rightarrow\delta}({x_i}^{\sigma_i}) \leftarrow \langle{\pi_i}^{\sigma_i}\rangle_\delta]$, for all $c^{\sigma_i\Rightarrow\delta}$ in $\mathit{Coerce}({x_i}^{\sigma_i}, {M_i}^{\tau_i})$.

By Inversion Lemma we have:

(1) $\Gamma \vdash \lambda {x_1}^{\sigma_1}.{M_1}^{\tau_1}|\cdots|\lambda {x_n}^{\sigma_n}.{M_n}^{\tau_n} : (\sigma_1 \to \tau_1) \wedge\cdots\wedge (\sigma_n \to \tau_n)$;

(2) $\Gamma \vdash {\pi_1}^{\sigma_1}|\cdots|{\pi_n}^{\sigma_n} : \sigma_1 \bigwedge\cdots\bigwedge \sigma_n$.

By Inversion Lemma on (1) we have:

(3) $\Gamma, z^\sigma : \sigma \vdash {\overline{M}_1}^{\tau_1}|\cdots|{\overline{M}_n}^{\tau_n} : \tau_1 \wedge\cdots\wedge \tau_n$

where

- $\sigma \sqsubseteq \sigma_i \sqsubseteq \mathrm{R}(\mathit{Coerce}(z^\sigma, {\overline{M}_i}^{\tau_i}))$, for all $i$, $1 \le i \le n$;
- ${\overline{M}_i}^{\tau_i} = {M_i}^{\tau_i}[c^{\sigma_i\Rightarrow\delta}({x_i}^{\sigma_i}) \leftarrow c^{\sigma\Rightarrow\delta}(z^\sigma)]$, for all $c^{\sigma_i\Rightarrow\delta}$ in $\mathit{Coerce}({x_i}^{\sigma_i}, {M_i}^{\tau_i})$.

Since $\sigma_i \sqsubseteq \mathrm{R}(\mathit{Coerce}(z^\sigma, {\overline{M}_i}^{\tau_i}))$, we can apply Lemma 31 to (3) and (2) and we obtain $\Gamma \vdash {\overline{N}_1}^{\tau_1}|\cdots|{\overline{N}_n}^{\tau_n} : \tau_1 \wedge\cdots\wedge \tau_n$, where ${\overline{N}_i}^{\tau_i} = {\overline{M}_i}^{\tau_i}[c^{\sigma\Rightarrow\delta}(z^\sigma) \leftarrow \langle{\pi_i}^{\sigma_i}\rangle_\delta]$, for all $c^{\sigma\Rightarrow\delta}$ in $\mathit{Coerce}(z^\sigma, {\overline{M}_i}^{\tau_i})$.

Since ${N_i}^{\tau_i} \equiv {\overline{N}_i}^{\tau_i}$ for all $i$ such that $1 \le i \le n$, we have the thesis.

Induction step. All cases follow easily from the induction hypothesis, by applying the generating rules. In particular, use Lemma 31 in the case of (App), similarly to the base case above. □

## 8. Conclusions and related approaches

In this paper we presented a typed lambda calculus which is an equivalent formulation à la Church of the well-known Intersection Type Discipline à la Curry (in which all strongly normalizing λ-terms have a typed version). The main ideas underlying our calculus can be summarized as follows: a relevant parallel term constructor represents the assignment of an intersection type to a pure λ-term; typed terms in parallel share the same set of typed variables, so that λ-abstraction can be defined as a simultaneous operation on all terms in parallel and acts on the same variable; by a local renaming, once a free typed variable is λ-abstracted, each of the terms in parallel can do its local renaming with relevant type refinement on that variable.

Now we summarize some related approaches, both from the point of view of the encoding of logical deductions for intersection types by lambda terms, and from the point of view of the definitions of intersection typed calculi independently from a logical characterization. We believe that the approaches considered in this section gave significant insights into the problem of finding a typed characterization of the Intersection Type Discipline (ITD), in that they all contributed, in different ways, to the main underlying ideas of our proposal.

Capitani, Loreti and Venneri introduced in [5] the notion of a *strict parallel* composition of deductions for representing the intersection derivability. They devised a propositional logic with hyperformulae and defined how to encode parallel deductions by untyped lambda terms in order to prove that the logical system of hyperformulae is isomorphic to ITD. However, the encoding proposed in [5] is partially a meta-level definition, since it involves transformations on the whole subdeductions, thus that approach cannot be extended to define correctly a typed lambda-abstraction.

Ronchi Della Rocca, in [27], discussed how to design an explicitly typed lambda calculus, corresponding to ITD, and based on the Intersection Logic IL [28]. Informally speaking, a deduction of IL is a set of intuitionistic deductions that can be thought of as writable in parallel. It is important to notice that a meta-level condition of structural equivalence on deductions is the key-point to decide how deductions can be grouped together in IL. Following this logical characterization, a complete decoration of IL deductions by typed terms is proposed in [27]. Indeed, the definition of this decorating algorithm involves the definition of the type erasure function on typed terms. Informally speaking, the meta-level condition of equivalence on IL deductions is here represented by "kits", which are trees whose leaves are labelled by pure lambda terms; thus checking that subterms coming from the same "kit" guarantees that their erasure yields the same untyped term. For these reasons, the resulting calculus is a decoration language which provides many suggestions towards a true typed calculus. The definition of "molecules", as a refinement of the notion of hyperformulae, has been formulated by Pimentel, Ronchi Della Rocca and Roversi in [24], in order to formalize how to group isomorphic deductions (type deductions for the same lambda term). In that paper, the authors obtain an elegant proof-theoretical account of ITD, which clarifies the main relations between the intuitionistic conjunction and the intersection operator. A decoration of such deductions is provided by using pure lambda terms that codify only the order of the application of the structural rules. Liquori and Ronchi Della Rocca presented a formulation à la Church of a lambda calculus with intersection types in a recent paper [18]. The novelty of this result, with respect to the approaches listed above, consists in proposing a calculus which is defined "per se" instead of decorating type deductions for untyped terms. However, this calculus is based on an unusual notion of context: each variable is associated with both a mark (an integer) and some types, in such a way that a typed variable is represented as a pair (variable, mark), where the mark can be associated with different intersection types. Thus, typed terms must always be associated to proof-trees representing different type derivations for different types of their marked variables.

Wells and Haack [36] proposed the first true explicitly typed lambda calculus with the same power as the Intersection Type Discipline. The notable characteristic of this proposal is the notion of "branching types" that are composed by a join operator to model their intersection. Namely, every well-formed type has a kind, in order to keep track of its branching shape. Then the key technical point is the operation of "expansion" that, applied to a type in a given branching shape, results in a new branching shape of that type where some of its subterms are duplicated. Comparing [36]'s proposal with our typed calculus, we can say that we adopt an opposite approach to the addressed problem. Branching types are devised to squash together different type deductions, concerning the same untyped lambda term, into a compact notation; in a sense, types can be viewed as type schemes representing all virtual instances that are obtained by expanding (duplicating) subdeductions according to the associated pattern (parameter). Completely different, our approach codifies all such expanded deductions by typed terms, that are constructed in a parallel composition by sharing the same resources and by using only synchronized rules for forming well-typed terms from well-typed subterms, where synchronization means that the rules act simultaneously on the components of a parallel term. From our point of view, a drawback of the calculus of [36] can be seen in the fact that typing rules can require to expand the type context in order to check the well-formedness of a typed term (see rule $\forall_i$ in [36]); as a consequence, also during typed beta-reduction, terms that are replaced by free variables must be expanded accordingly. Instead, our goal was to define a typed calculus whose terms are simply generated by their typed subterms, without involving any operation on free variables, in such a way that an Inversion Lemma naturally arises. We think we have reached such a goal.

## Acknowledgements

This work is meant as a tribute and homage to Mariangiola, Mario and Simona, who invented and developed the intersection type discipline. In particular, Betti and Viviana wholeheartedly thank them for being invaluable guides in their introduction to type theory and to scientific research, and for honoring them with their precious scientific cooperation as well as their friendship.

The authors would like to thank the anonymous referees for their comments which helped in greatly improving the presentation of the paper.

## References

[1] Fabio Alessi, Franco Barbanera, Strong conjunction and intersection types, in: A. Tarlecki (Ed.), MFCS'91, in: LNCS, vol. 520, Springer, 1991, pp. 64–73.
[2] Fabio Alessi, Franco Barbanera, Mariangiola Dezani-Ciancaglini, Intersection types and lambda models, Theoretical Compututer Science 355 (2) (2006) 108–126.
[3] Henk Barendregt, Mario Coppo, Mariangiola Dezani-Ciancaglini, A filter lambda model and the completeness of type assignment, The Journal of Symbolic Logic 48 (4) (1983) 931–940.
[4] Martin W. Bunder, Intersection types for lambda-terms and combinators and their logics, Logic Journal of the IGPL 10 (4) (2002) 357–378.
[5] Beatrice Capitani, Michele Loreti, Betti Venneri, Hyperformulae, parallel deductions and intersection types, in: J.-J. Lévy (Ed.), Proc. of BOTH 2001, in: ENTCS, vol. 50, Elsevier, 2001, pp. 180–198.
[6] Mario Coppo, Mariangiola Dezani-Ciancaglini, A new type assignment for λ-terms, Archiv für mathematische Logik und Grundlagenforschung 19 (3–4) (1978) 139–156.
[7] Mario Coppo, Mariangiola Dezani-Ciancaglini, Patrick Sallé, Functional characterization of some semantic equalities inside λ-calculus, in: H.A. Maurer (Ed.), Proc. of Colloquium on Automata, Languages and Programming, in: LNCS, vol. 71, Springer, 1979, pp. 133–146.
[8] Mario Coppo, Mariangiola Dezani-Ciancaglini, Betti Venneri, Principal type schemes and λ-calculus semantics, in: Seldin and Hindley [29], pp. 535–560.
[9] Mariangiola Dezani-Ciancaglini, Alain Frisch, Elio Giovannetti, Yoko Motohama, The relevance of semantic subtyping, in: S. van Bakel (Ed.), Proc. of ITRS 2002, in: ENTCS, vol. 70, 2002.
[10] Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Betti Venneri, The relevance of intersection and union types, Notre Dame Journal of Formal Logic 38 (2) (1997) 246–269.
[11] Mariangiola Dezani-Ciancaglini, Robert K. Meyer, Yoko Motohama, The semantics of entailment Omega, Notre Dame Journal of Formal Logic 43 (3) (2002) 129–145.
[12] Alain Frisch, Giuseppe Castagna, Véronique Benzaken, Semantic subtyping, in: Proc. of the Symposium on Logic in Comput. Science, LICS'02, IEEE Computer Society, 2002, pp. 137–146.
[13] Jean-Yves Girard, Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur, Thèse d'état, University of Paris VII, 1972. Summary in J.E. Fenstad (Ed.), Scandinavian Logic Symposium, North-Holland, 1971, pp. 63–92.
[14] Jean-Yves Girard, Paul Taylor, Yves Lafont, Proofs and Types, in: Cambridge Tracts in Theoretical Computer Science, vol. 7, Cambridge University Press, 1989.
[15] J. Roger Hindley, Basic Simple Type Theory, in: Cambridge Tracts in Theoretical Computer Science, vol. 42, Cambridge University Press, 1997.
[16] William A. Howard, The formulas-as-types notion of construction: in, Seldin and Hindley [29], pp. 479–490.
[17] Assaf J. Kfoury, Joe B. Wells, Principality and decidable type inference for finite-rank intersection types, in: Proc. of Principles of Programming Languages, POPL'99, ACM, 1999, pp. 161–174.
[18] Luigi Liquori, Simona Ronchi della Rocca, Intersection-types à la Church, Information and Computation 205 (9) (2007) 1371–1386.
[19] E.K.G. Lopez-Escobar, Proof-functional connectives, in: C. Di Prisco (Ed.), Methods of Mathematical Logic, (Proc. of Latin-American Symposium on Mathematical Logic), in: LNCS, vol. 1130, Springer, 1985, pp. 208–221.
[20] Robert K. Meyer, Richard Routley, Algebraic analysis of entailment I, Logique et Analyse 15 (1972) 407–428.
[21] Robert K. Meyer, Richard Routley, Algebraic analysis of entailment III, Journal of Philosophical Logic 1 (1972) 192–208.
[22] John C. Mitchell, Foundations for Programming Languages, The MIT Press, 1996.
[23] Benjamin C. Pierce, Programming with Intersection Types and Bounded Polymorphism, Ph.D. Thesis, Carnegie Mellon University, 1991.
[24] Elaine Pimentel, Simona Ronchi Della Rocca, Luca Roversi, Intersection types: A proof-theoretical approach: in, P. Bruscoli, F. Lamarche, C. Stewart, (Eds.), Proc. of Structures and Deduction — ICALP'05 Workshop, 16–17 July, 2005.
[25] John C. Reynolds (Eds.), Towards a theory of type structure, in: B. Robinet (Ed.), Programming Symposium, in: LNCS, vol. 19, Springer, 1974, pp. 408–425.
[26] John C. Reynolds, Design of the programming language Forsythe, in: P. O'Hearn, R.D. Tennent (Eds.), Algol-like Languages, Birkhauser, 1996.
[27] Simona Ronchi Della Rocca, Typed intersection lambda calculus, in: S. van Bakel (Ed.), Proc. of ITRS 2002, in: ENTCS, vol. 70, Elsevier, 2002.
[28] Simona Ronchi Della Rocca, Luca Roversi, Intersection logic, in: L. Fribourg (Ed.), Proc. of CSL'01, in: LNCS, vol. 2412, Springer, 2001, pp. 414–428.
[29] Jonathan P. Seldin, J. Roger Hindley (Eds.), To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press, 1980.
[30] Morten H. Sørensen, Pawel Urzyczyn, Lectures on the Curry–Howard Isomorphism, in: Studies in Logic and the Foundations of Mathematics, vol. 149, Elsevier, 2006.
[31] Pawel Urzyczyn, Type reconstruction in $F_\omega$, Mathematical Structures in Computer Science 7 (4) (1997) 329–358.

[32] Steffen van Bakel, Intersection type assignment systems, Theoretical Computer Science 151 (2) (1995) 385–435.
[33] Steffen van Bakel, Cut-Elimination in the strict intersection type assignment system is strongly normalizing, Notre Dame Journal of Formal Logic 45 (1) (2004) 35–63.
[34] Betti Venneri, Intersection types as logical formulae, Journal of Logic and Computation 4 (2) (1994) 109–124.
[35] Joe B. Wells, Allyn Dimock, Robert Muller, Franklyn Turbak, A typed intermediate language for flow-directed compilation, in: M. Bidoit, M. Dauchet (Eds.), Proc. of TAPSOFT'97, in: LNCS, vol. 1214, Springer, 1997, pp. 757–771.
[36] Joe B. Wells, Christian Haack, Branching types, in: D. Le Métayer (Ed.), Proc. of ESOP'02, in: LNCS, vol. 2305, Springer, 2002, pp. 115–132.