Intersection-types à la Church

Luigi Liquori, Simona Ronchi Della Rocca

▶ **To cite this version:**

Luigi Liquori, Simona Ronchi Della Rocca. Intersection-types à la Church. Information and Computation, 2007, 205 (9), pp.1371-1386. 10.1016/j.ic.2007.03.005 . hal-01148282

HAL Id: hal-01148282
https://inria.hal.science/hal-01148282

Submitted on 4 May 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Intersection-Types *à la* Church

Luigi Liquori

INRIA Sophia Antipolis, France

Simona Ronchi Della Rocca

Dipartimento di Informatica, Università di Torino, Italy

Abstract

In this paper, we present $\Lambda^{\mathsf{t}}_{\wedge}$, a fully typed λ-calculus based on the intersection-type system discipline, which is a counterpart *à la* Church of the type assignment system as invented by Coppo and Dezani. The relationship between $\Lambda^{\mathsf{t}}_{\wedge}$ and the intersection type assignment system is the standard isomorphism between typed and type assignment system, and so the typed language inherits from the untyped system all the good properties, like subject reduction and strong normalization. Moreover both type checking and type reconstruction are decidable.

Key words: Logics and Intersection-Types, λ-calculus *à la* Curry and *à la* Church

1 Introduction

The implicative and conjunctive fragment of the intuitionistic logic (denoted by $\mathcal{L}^{\wedge}_{\rightarrow}$), where the logical connectives "$\rightarrow$" and "$\wedge$" denote the implication and the conjunction is a well-known powerful logical system: this logic is presented in Figure 1. Finding a typed λ-calculus *à la* Church, in the Curry-Howard sense is not a simple task [Hindley (1984)], because of the "anomalous" decoration of the rules dealing with conjunction.

The *Intersection-Type Assignment System* ($\vdash_{\wedge}$) is a set of inference rules for assigning *intersection-types* to terms of the untyped λ-calculus. Intersection-types are formulæ of the implicational and conjunctive fragment of propositional logic. The syntax and the typing rules are presented in Figure 2. Intersection-

Email addresses: Luigi.Liquori@inria.fr (Luigi Liquori), ronchi@di.unito.it (Simona Ronchi Della Rocca).

Let $\Sigma \triangleq \{\sigma_1, \ldots, \sigma_n\}$, and $\Sigma, \sigma \triangleq \Sigma \cup \{\sigma\}$

$$\frac{\sigma \in \Sigma}{\Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma}\,(Var) \qquad \frac{\Sigma, \sigma_1 \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_2}{\Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_1 \rightarrow \sigma_2}\,(\rightarrow I)$$

$$\frac{\Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_1 \rightarrow \sigma_2 \quad \Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_1}{\Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_2}\,(\rightarrow E) \qquad \frac{\Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_1 \quad \Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_2}{\Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_1 \wedge \sigma_2}\,(\wedge I)$$

$$\frac{\Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_1 \wedge \sigma_2}{\Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_1}\,(\wedge E_L) \qquad \frac{\Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_1 \wedge \sigma_2}{\Sigma \vdash_{\mathcal{L}^{\wedge}_{\rightarrow}} \sigma_2}\,(\wedge E_R)$$

Figure 1. The Logic $\mathcal{L}^{\wedge}_{\rightarrow}$

Syntax of $\Lambda^{\mathsf{u}}_{\wedge}$
Let α range over a denumerable set $\mathcal{V}$ of type-constants
Let "$\wedge$" take precedence over "$\rightarrow$"

$$M ::= x \mid \lambda x.M \mid M\,M$$

$$\sigma ::= \alpha \mid \sigma \rightarrow \sigma \mid \sigma \wedge \sigma$$

Type System for $\Lambda^{\mathsf{u}}_{\wedge}$
Let $E \triangleq \{x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n\}$ ($i \neq j$ implies $x_i \not\equiv x_j$), and $E, x{:}\sigma \triangleq E \cup \{x{:}\sigma\}$

$$\frac{x{:}\sigma \in E}{E \vdash_{\wedge} x : \sigma}\,(Var) \qquad \frac{E, x{:}\sigma_1 \vdash_{\wedge} M : \sigma_2}{E \vdash_{\wedge} \lambda x.M : \sigma_1 \rightarrow \sigma_2}\,(\rightarrow I)$$

$$\frac{E \vdash_{\wedge} M : \sigma_1 \rightarrow \sigma_2 \quad E \vdash_{\wedge} N : \sigma_1}{E \vdash_{\wedge} M\,N : \sigma_2}\,(\rightarrow E) \qquad \frac{E \vdash_{\wedge} M : \sigma_1 \quad E \vdash_{\wedge} M : \sigma_2}{E \vdash_{\wedge} M : \sigma_1 \wedge \sigma_2}\,(\wedge I)$$

$$\frac{E \vdash_{\wedge} M : \sigma_1 \wedge \sigma_2}{E \vdash_{\wedge} M : \sigma_1}\,(\wedge E_L) \qquad \frac{E \vdash_{\wedge} M : \sigma_1 \wedge \sigma_2}{E \vdash_{\wedge} M : \sigma_2}\,(\wedge E_R)$$

Figure 2. The Intersection-Type Assignment System $\vdash_{\wedge}$.

types were introduced by Coppo and Dezani, to increase the typability power of Curry's type assignment system for the λ-calculus [Coppo and Dezani-Ciancaglini (1980)]. Since then, intersection-types have been fruitfully used for designing static semantics of programming languages (*e.g.* Algol-like [Reynolds (1996)]), for characterizing interesting classes of λ-terms (*e.g.* the strongly normalizing ones [Pottinger (1980)]), and for studying denotational semantics of various untyped λ-calculi (*e.g.* [Barendregt et al. (1983)] and [Coppo et al. (1983)]).

There are many versions in the literature of intersection-type assignment systems. Here we choose that one presented as "System D" [Krivine (1990)], characterized

by the presence of non syntax-directed rules for dealing with the introduction and elimination of the intersection. Note that, differently from most of the systems presented in the literature, as for example [Dezani-Ciancaglini et al. (1998)], in this system the connective $\wedge$ is neither commutative nor associative nor idempotent. The choice of this presentation has been taken since we are looking for a typed version of the calculus, where bound variables come decorated with their types, and in this setting it is natural to consider types as syntactical entities. In any case, this presentation does not have any consequence on the typability power of the intersection type assignment system, which is well known to characterize all and only the strongly normalizing terms [Krivine (1990), Pottinger (1980)].

Following the standard terminology, let us call *à la* Curry a system assigning types to untyped terms, and *à la* Church a system assigning types to typed terms, *i.e.* where types are part of the syntax of terms, by decorating bound-variables in abstractions. Differently from other type assignment systems *à la* Curry, $\Lambda^{\mathsf{u}}_{\wedge}$ has no natural counterpart *à la* Church. The classical example is the polymorphic identity in $\Lambda^{\mathsf{u}}_{\wedge}$ that has the following type-derivation:

$$\dfrac{\dfrac{x{:}\sigma_1 \vdash_\wedge x : \sigma_1}{\vdash_\wedge \lambda x.x : \sigma_1{\to}\sigma_1}(\to I) \quad \dfrac{x{:}\sigma_2 \vdash_\wedge x : \sigma_2}{\vdash_\wedge \lambda x.x : \sigma_2{\to}\sigma_2}(\to I)}{\vdash_\wedge \lambda x.x : (\sigma_1{\to}\sigma_1){\wedge}(\sigma_2{\to}\sigma_2)}(\wedge I)$$

but is untypable using a naïve corresponding rule *à la* Church for the introduction of intersection-types [Hindley (1984)].

$$\dfrac{\dfrac{x{:}\sigma_1 \vdash_\wedge x : \sigma_1}{\vdash_\wedge \lambda x{:}\sigma_1.x : \sigma_1{\to}\sigma_1}(\to I) \quad \dfrac{x{:}\sigma_2 \vdash_\wedge x : \sigma_2}{\vdash_\wedge \lambda x{:}\sigma_2.x : \sigma_2{\to}\sigma_2}(\to I)}{\vdash_\wedge \lambda x{:}\boxed{\ ?\ }.x : (\sigma_1{\to}\sigma_1){\wedge}(\sigma_2{\to}\sigma_2)}(\wedge I)$$

By the Curry-Howard isomorphism [Howard (1980)], a λ-term must record the shape of its type-derivation. A standard proof decoration would give rise to a language which is a λ-calculus extended with a pair construction. For example, according to [Ronchi Della Rocca (2002)], the previous proof would be decorated in the following way:

$$\dfrac{\dfrac{x{:}\sigma_1 \vdash_\wedge x : \sigma_1}{\vdash_\wedge \lambda x{:}\sigma_1.x : \sigma_1{\to}\sigma_1}(\to I) \quad \dfrac{x{:}\sigma_2 \vdash_\wedge x : \sigma_2}{\vdash_\wedge \lambda x{:}\sigma_2.x : \sigma_2{\to}\sigma_2}(\to I)}{\vdash_\wedge < \lambda x{:}\sigma_1.x, \lambda x{:}\sigma_2.x > : (\sigma_1{\to}\sigma_1){\wedge}(\sigma_2{\to}\sigma_2)}(\wedge I)$$

The resulting language has a huge syntax, since the pairing construct can be applied only on terms, which can be different, but their below untyped versions must be identical. An example in the literature of λ-calculus typed *à la* Church with intersection types, where the syntax is exactly the classical one, but for types,

is the language Forsythe in [Reynolds (1996)]. But it is incomplete, in the sense that the resulting typed system has less typability power than the type assignment one. In fact, in the Reynolds syntax, assuming that a term M has type τ under the assumption that the variable x has any one of the types σ_i $(1{\leq}i{\leq}n)$, we can form the typed term:

$$\lambda x{:}\sigma_1|\sigma_2|...|\sigma_n.M$$

having types $(\sigma_i{\to}\tau)$ for $1{\leq}i{\leq}n$, and all types derived from these by applying intersection introduction, intersection elimination and subtyping relations. So, for example, there is not a typed version of $\lambda x.\lambda y.x$, giving it the type $\rho = (\sigma{\to}(\sigma{\to}\sigma)) \wedge (\tau{\to}(\tau{\to}\tau))$, where σ and τ are uncomparable. In fact, according to the Forsythe syntax, we can form the two terms, namely $\lambda x{:}\sigma|\tau.\lambda y{:}\sigma.x$, having types $\sigma{\to}(\sigma{\to}\sigma)$ and $\tau{\to}(\sigma{\to}\tau)$, and $\lambda x{:}\sigma|\tau.\lambda y{:}\tau.x$, having types $\tau{\to}(\tau{\to}\tau)$ and $\sigma{\to}(\tau{\to}\sigma)$, but there is not a term relating the types of x and y in the desired way. (this example has been taken from [Wells and Haack (2006)]). Moreover, Forsythe appear something in between a typed and a type assignment calculus, since terms do not have unique types.

The problem is, as the skilled reader can understand, the presence of *non syntax-directed rules* that disconnect the λ-term from its type-derivation (hence losing the Curry-Howard correspondence). It is important to point out that this problem does not depend on the chosen intersection-type assignment system; indeed, not one of the intersection type assignment systems presented in the literature is completely syntax directed (and not-even it cannot be!)

Our goal is to build a λ-calculus *à la* Church, and related intersection-type system $\Lambda^{\mathsf{t}}_{\wedge}$, whose syntax is, as far as possible, similar to other typed λ-calculi. We want to design this calculus through a typed system, building typed terms together with their type, such that the typed system and the type assignment system $\Lambda^{\mathsf{u}}_{\wedge}$ are related by the standard path designed in [Giannini et al. (1993), Liquori (1996), van Bakel et al. (1997)]. For this system, we are interested to catch as much as possible properties from the following list:

Desiderata

(1) typed and type assignment derivations are *isomorphic*, under the assumption that they share the same type syntax. *I.e.*, the application of an *erasing function* $\mathcal{E}$ on all typed terms and contexts (in a typed derivation judgment) produces a derivable type assignment derivation with the same structure, and every type assignment derivation is obtained from a typed one with the same structure by applying the same erasure $\mathcal{E}$. Such a kind of isomorphism has been studied in [van Bakel et al. (1997)].

Moreover, we want that the intersection calculus *à la* Church inherits all the properties of intersection type assignment *à la* Curry, namely:

(2) subject reduction;

(3) strong normalization of typable terms;

plus the following ones:

(4) unicity of typing;
(5) decidability of type reconstruction and of type checking.

In order to find a solution to this challenge, we designed a calculus, whose terms are composed by two parts, carrying out the computational and logical information respectively. The first component (the *marked-term*) is a simply typed λ-term, but types are variable-marks, *i.e.*, natural numbers representing store locations. The second component (the *proof-term*) records both the associations between variable-marks and types and the structure of the derivation. The technical tool for realizing this is an unusual formulation of context, which assigning types to term-variables *at a given mark/location*. The calculus of proof-terms can be defined *per se*, as decoration of the implicative and conjunctive fragment of intuitionistic logic; it codifies a set of proofs that is strictly bigger than these corresponding to intersection-type derivations (see [Ronchi Della Rocca and Roversi (2001)]).

As example, the typed identity with type

$$(\sigma_1 \rightarrow \sigma_1) \wedge (\sigma_2 \rightarrow \sigma_2)$$

can be written in our proposal as the term

$$(\lambda x{:}0.x)@(\lambda 0{:}\sigma_1.0) \wedge (\lambda 0{:}\sigma_2.0)$$

where 0 is a mark, and $(\lambda 0{:}\sigma_1.0) \wedge (\lambda 0{:}\sigma_2.0)$ is the logical content of $\lambda x{:}0.x$. The typed λ-calculus so obtained satisfies all the above requirements. As a nice consequence of these choices, we get decidability of the type reconstruction and type checking, both being also easy to define.

There are other proposals in the literature for a λ-calculus typed with intersection types. We already have cited Forsythe [Reynolds (1996)], which is not complete, as well as the language proposed in [Pierce and Turner (1994)]. The languages proposed in [Capitani and Venneri (2001)], [Wells et al. (2002)], and [Wells and Haack (2006)] have been designed with other purposes, and they do not satisfy requirement 1. The language in [Ronchi Della Rocca (2002)] has been designed for logical purposes in order to satisfy the requirement 1, but its syntax and operational semantics are unsatisfying, from our point of view.

The paper is organized as follows: Section 2 presents the logical calculus. Section 3 shows the whole intersection-typed λ-calculus. Section 4 contains some examples, while Section 5 lists the metatheory and the type checking/type reconstruction algorithms. In Section 6 the soundness between $\Lambda^{\mathsf{t}}_{\wedge}$ and $\Lambda^{\mathsf{u}}_{\wedge}$ is proved. Conclusions and final remarks end the paper.

2 The Proof-calculus $\Lambda\mathcal{P}$

The syntax of intersection-types is that of the formulas of the implicative and conjunctive fragment of the intuitionistic logic (denoted by $\mathcal{L}^{\wedge}_{\rightarrow}$), where the logical connectives "$\rightarrow$" and "$\wedge$" denote the implication and the conjunction. Unfortunately, the intersection-type assignment system $\Lambda^{\mathsf{u}}_{\wedge}$ does not correspond, in the Curry-Howard sense, to this logic [Hindley (1984)], because of the "anomalous" decoration of the rules dealing with conjunction.

In what follows, we present a typed λ-calculus, obtained by decorating the proof of such a logic. The main peculiarity of this calculus is that it is defined on another categories of variables called *variable-marks*; the calculus will be used to record the structure of an intersection derivation, though an association between variable-marks and types.

Syntax of $\Lambda\mathcal{P}$. We start with some useful definitions.

Definition 2.1

(1) Variable-marks (denoted by ι) range over Nat*;*
(2) Intersection-types are defined as follows:

$$\sigma ::= \alpha \mid \sigma{\rightarrow}\sigma \mid \sigma \wedge \sigma$$

where α ranges over a denumerable set $\mathcal{V}$ of constants;
(3) Proof-contexts are finite associations between variable-marks and types, and they are defined by the following grammar:

$$G ::= \epsilon \mid G, \iota{:}\sigma$$

(4) Pseudo-proof-trees are labeled unary/binary trees defined as follows:

$$\Delta ::= \iota \mid \lambda\iota{:}\sigma.\Delta \mid \Delta\,\Delta \mid \Delta{\wedge}\Delta \mid {\swarrow}\Delta \mid \Delta{\searrow}$$

(5) The set $\mathsf{Fv}(\Delta)$ *of the variable-marks in a pseudo-proof-tree Δ is*

$$\mathsf{Fv}(\iota) \triangleq \{\iota\} \qquad \mathsf{Fv}(\lambda\iota{:}\sigma.\Delta) \triangleq \mathsf{Fv}(\Delta) \setminus \{\iota\}$$

$$\mathsf{Fv}(\Delta{\searrow}) \triangleq \mathsf{Fv}(\Delta) \qquad \mathsf{Fv}(\Delta_1\,\Delta_2) \triangleq \mathsf{Fv}(\Delta_1) \cup \mathsf{Fv}(\Delta_2)$$

$$\mathsf{Fv}({\swarrow}\Delta) \triangleq \mathsf{Fv}(\Delta) \qquad \mathsf{Fv}(\Delta_1{\wedge}\Delta_2) \triangleq \mathsf{Fv}(\Delta_1) \cup \mathsf{Fv}(\Delta_2)$$

A variable-mark is bound in Δ if it is not free in Δ.

The Proof-calculus $\Lambda\mathcal{P}$.
Let $G \triangleq \{\iota_1{:}\sigma_1, \ldots, \iota_n{:}\sigma_n\}$ ($i \neq j$ implies $\iota_i \not\equiv \iota_j$), and $G, \iota{:}\sigma \triangleq G \cup \{\iota{:}\sigma\}$

$$\frac{\iota{:}\sigma \in G}{G \vdash_P \iota : \sigma}\,(Var) \qquad \frac{G, \iota{:}\sigma_1 \vdash_P \Delta : \sigma_2}{G \vdash_P \lambda\iota{:}\sigma_1.\Delta : \sigma_1{\to}\sigma_2}\,(\to I)$$

$$\frac{G \vdash_P \Delta_1 : \sigma_1{\to}\sigma_2 \quad G \vdash_P \Delta_2 : \sigma_1}{G \vdash_P \Delta_1\,\Delta_2 : \sigma_2}\,(\to E) \qquad \frac{G \vdash_P \Delta_1 : \sigma_1 \quad G \vdash_P \Delta_2 : \sigma_2}{G \vdash_P \Delta_1{\wedge}\Delta_2 : \sigma_1 \wedge \sigma_2}\,(\wedge I)$$

$$\frac{G \vdash_P \Delta : \sigma_1 \wedge \sigma_2}{G \vdash_P \swarrow\!\Delta : \sigma_1}\,(\wedge E_L) \qquad \frac{G \vdash_P \Delta : \sigma_1 \wedge \sigma_2}{G \vdash_P \Delta\!\searrow : \sigma_2}\,(\wedge E_R)$$

Figure 3. The Proof-calculus $\Lambda\mathcal{P}$.

Type System for $\Lambda\mathcal{P}$. The system proves judgments of the shape:

$$G \vdash_P \Delta : \sigma$$

where G is a proof-context, Δ is a pseudo-proof-tree, and σ is a type. The pseudo-proof-tree Δ is a *legal* proof-tree if there are G and σ such that $G \vdash_P \Delta : \sigma$. The rules of the system, obtained by decorating the rules of the logic $\mathcal{L}^{\wedge}_{\to}$, are showed in Figure 2. Note that $\Lambda\mathcal{P}$ is just an unusual syntax for the simply typed λ-calculus with pairs, which can be seen, via the Curry-Howard isomorphism, as a decoration of $\mathcal{L}^{\wedge}_{\to}$.

Reduction Semantics of $\Lambda\mathcal{P}$. Being $\Lambda\mathcal{P}$ a calculus isomorphic to the typed λ-calculus with pairs, its reduction rules are the well known ones:

$$(\lambda\iota{:}\sigma.\Delta_1)\,\Delta_2 \;\to_\beta\; \Delta_1[\Delta_2/\iota]$$

$$\swarrow\!(\Delta_1{\wedge}\Delta_2) \;\to_{\pi_1}\; \Delta_1$$

$$(\Delta_1{\wedge}\Delta_2)\!\searrow \;\to_{\pi_2}\; \Delta_2$$

By abuse of notation, $\to_\iota$, $\to_{\pi_1}$, $\to_{\pi_2}$ will denote the contextual closure of the above rules.

As usual, the $\Lambda\mathcal{P}$ calculus works modulo α-conversion, as the symmetric, transitive, reflexive, and contextual closure of the following rule:

$$\lambda\iota_1{:}\sigma.\Delta \to_\alpha \lambda\iota_2{:}\sigma.\Delta[\iota_2/\iota_1] \qquad \text{where } \iota_2 \text{ is fresh}$$

The following result holds:

Fact 1 (Strong Normalization of $\Lambda\mathcal{P}$) *$\Lambda\mathcal{P}$ is strongly normalizing.*

3 The Intersection-Typed Calculus $\Lambda^{\mathsf{t}}_{\wedge}$

The key idea in the design of the intersection-typed system is to split the term into two parts, carrying out the computational and the logical information respectively. Namely, the first one is a term of a typed λ-calculus, while the second one is a proof-term (belonging to the language $\Lambda\mathcal{P}$ introduced in the previous section) describing the shape of the type derivation. The technical tool for connecting the two parts is an unusual formulation of contexts. In fact, a context associates to a variable both a variable-mark *and* a type, such that different variables are associated to different variable-marks.

This novel formulation of contexts allows to remember, in rule $(\rightarrow I)$, just the variable-mark, the corresponding type being stored in the proof-tree, built by the system *in parallel* with the typed term. In this way the underlying term is *de facto* a term of the classical untyped λ-calculus. Since the proof-tree describes the structure of the type-derivation, we also obtain the decidability of type reconstruction and type checking.

Syntax.

Definition 3.1

(1) Contexts are finite associations between variables and types at a given variable-mark, such that each variable and each variable-mark occur at least once in it. They are defined as follows:

$$\Gamma ::= \epsilon \mid \Gamma, x@\iota{:}\sigma$$

(2) Marked-terms are defined as follows:

$$M ::= x \mid \lambda x{:}\iota.M \mid M\ M$$

(3) A pseudo-term of $\Lambda^{\mathsf{t}}_{\wedge}$ has the shape $M@\Delta$, where M is a marked-term and Δ is a proof-tree;

In what follows, the symbol $\equiv$ denotes the syntactic equality for marked-terms, types, contexts, variable-marks and proof-trees, respectively.

Definition 3.2 (Fv)

(1) The set of free-variables of a marked-term is defined as follows:

$$\mathsf{Fv}(x) \triangleq \{x\}\ \mathsf{Fv}((\lambda x{:}\iota.M)) \triangleq \mathsf{Fv}(M) \setminus \{x\}\ \mathsf{Fv}((M\ N)) \triangleq \mathsf{Fv}(M) \cup \mathsf{Fv}(N)$$

$$\frac{x@\iota{:}\sigma \in \Gamma}{\Gamma \vdash x@\iota : \sigma}\,(Var) \qquad \frac{\Gamma, x@\iota{:}\sigma_1 \vdash M@\Delta : \sigma_2}{\Gamma \vdash (\lambda x{:}\iota.M)@(\lambda\iota{:}\sigma_1.\Delta) : \sigma_1 \to \sigma_2}\,(\to I)$$

$$\frac{\Gamma \vdash M@\Delta_1 : \sigma_1 \to \sigma_2 \quad \Gamma \vdash N@\Delta_2 : \sigma_1}{\Gamma \vdash (M\,N)@(\Delta_1\,\Delta_2) : \sigma_2}\,(\to E) \qquad \frac{\Gamma \vdash M@\Delta_1 : \sigma_1 \quad \Gamma \vdash M@\Delta_2 : \sigma_2}{\Gamma \vdash M@(\Delta_1 \wedge \Delta_2) : \sigma_1 \wedge \sigma_2}\,(\wedge I)$$

$$\frac{\Gamma \vdash M@\Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\swarrow\!\Delta) : \sigma_1}\,(\wedge E_L) \qquad \frac{\Gamma \vdash M@\Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\Delta\!\searrow) : \sigma_2}\,(\wedge E_R)$$

Figure 4. The Type System $\vdash$ for $\Lambda^{\mathsf{t}}_{\wedge}$.

(2) The set of free variables and of free variable-marks of a pseudo term is defined as follows:

$$\mathsf{Fv}(M@\Delta) \triangleq \mathsf{Fv}(M) \cup \mathsf{Fv}(\Delta)$$

Type System $\vdash$. The type judgments of the intersection-typed calculus $\Lambda^{\mathsf{t}}_{\wedge}$ have the shape:

$$\Gamma \vdash M@\Delta : \sigma$$

where Γ is a context, M is a marked-term, and Δ is a proof-tree. Intuitively: in the judgment, the type-context Γ assigns intersection-types to the free-variables of M annotated by variable-marks; if $\Gamma \vdash M@\Delta : \sigma$, then we say that $M@\Delta$ is a term of $\Lambda^{\mathsf{t}}_{\wedge}$.

The proof-tree keeps track of the type of the used mark together with a trace of the *skeleton* of the derivation tree. The proof-tree Δ plays the role of a road map to backtrack (*i.e.* roll back) the derivation tree. The typing rules are presented in Figure 3. Some comments are in order:

- (Var) gives types to free-variables at a given mark;
- $(\to I)$ is a quasi-classical abstraction rule, but it records in the term only the variable-mark associated to the abstracted variable; the proof-tree Δ evolves in a new proof-tree enriched with the binding for the mark ι;
- $(\to E)$ is a quasi-classical application rule; observe that the two type-stores of the premises become sub proof-trees in the conclusion (the hidden application operator being the root);
- $(\wedge I)$ is the most important rule; given two judgments for M assigning types σ_1 and type σ_2, in the same context Γ but with different proof-trees Δ_1, and Δ_2, we can assign the intersection-type $\sigma_1 \wedge \sigma_2$ to M in the context Γ but in the new proof-tree $\Delta_1 \wedge \Delta_2$. At this point the marked-term M *loses the one-to-one correspondence with its proof*. Luckily, the new proof-tree keeps track of the derivation and guarantees unicity of typing;
- $(\wedge E_L)$, and $(\wedge E_R)$ are the two standard rules that eliminate intersection-types. Also in this case the marked-term M *loses the one-to-one correspondence with*

its (logical) proof, but the proof is memorized by the proof-tree, thanks to the two place-holders $\swarrow$ and $\searrow$, indicating the applied rule.

Reduction semantics. For a given term $M@\Delta$, the computational part (M) and the logical part (Δ) grow up together while they are built through application of rules (Var), ($\rightarrow I$), and ($\rightarrow E$), but they *get disconnected* when we apply ($\wedge\{I, E_L, E_R\}$) rules (that changes the Δ but not the M). This disconnection is "logged" in the Δ via occurrences of operators $\wedge, \searrow$, and $\swarrow$.

As such:

- by $(\wedge I)$ and from $M@\Delta_1$ and $M@\Delta_2$, we get $M@(\Delta_1 \wedge \Delta_2)$. So Δ_1 and Δ_2 describe two different derivations sharing the same marked term. As a consequence, to a redex in M will correspond some redexes in Δ_1 and Δ_2, that need to be performed in parallel with the redex in M in order to preserve the correct syntax of the term;
- by $(\wedge E_L)$ or $(\wedge E_R)$ and from $M@\Delta$, we get $M@(\swarrow\Delta)$ or $M@(\Delta\searrow)$; so each reduction in M implies some reductions inside the wrapped Δ.

In order to correctly identify the reductions that need to be performed in parallel in the two parts of the term, we will define the notion of overlapping. Namely a redex is defining taking into account the surrounding context.

Definition 3.3 (Reduction Semantics)

(Contexts and Overlapping) *The marked-term contexts* $\mathsf{C}\{\,\}$ *and proof-term context* $\mathsf{D}\{\,\}$ *and multi-hole proof-term context* $\mathsf{E}\{\,\}$ *are defined as follows:*

$$\mathsf{C}\{\,\} ::= \{\,\} \mid \mathsf{C}\{\,\}\, M \mid M\, \mathsf{C}\{\,\} \mid \lambda x{:}\iota.\mathsf{C}\{\,\}$$

$$\mathsf{D}\{\,\} ::= \{\,\} \mid \mathsf{D}\{\,\}\, \Delta \mid \Delta\, \mathsf{D}\{\,\} \mid \lambda \iota{:}\sigma.\mathsf{D}\{\,\} \mid \swarrow \mathsf{D}\{\,\} \mid \mathsf{D}\{\,\} \searrow \mid \mathsf{D}\{\,\} \wedge \Delta \mid \Delta \wedge \mathsf{D}\{\,\}$$

$$\mathsf{E}\{\,\} ::= \{\,\} \mid \mathsf{E}\{\,\}\, \Delta \mid \Delta\, \mathsf{E}\{\,\} \mid \lambda \iota{:}\sigma.\mathsf{E}\{\,\} \mid \swarrow \mathsf{E}\{\,\} \mid \mathsf{E}\{\,\} \searrow \mid \mathsf{E}\{\,\} \wedge \mathsf{E}\{\,\}$$

Note that, while $\mathsf{C}\{\,\}$ *and* $\mathsf{D}\{\,\}$ *are contexts with exactly one hole, the context* $\mathsf{E}\{\,\}$ *can have more than one hole. Let* $\mathsf{E}\{\Delta_i\}_{i\in I}$ *denote a proof-context where the i-th hole has been filled by* Δ_i*, for* $i \in I$*. The notion of overlapping between marked-terms and proof-terms, denoted by* $\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}\{\,\}$ *is defined by induction on* $\mathsf{E}\{\,\}$ *as follows:*

$$\frac{}{\{\,\} \sqsubseteq \{\,\}} \qquad \frac{\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}\{\,\}}{\mathsf{C}\{\,\}\, M \sqsubseteq \mathsf{E}\{\,\}\, \Delta} \qquad \frac{\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}\{\,\}}{M\, \mathsf{C}\{\,\} \sqsubseteq \Delta\, \mathsf{E}\{\,\}} \qquad \frac{\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}\{\,\}}{\lambda x{:}\iota.\mathsf{C}\{\,\} \sqsubseteq \lambda \iota{:}\sigma.\mathsf{E}\{\,\}}$$

$$\frac{\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}\{\,\}}{\mathsf{C}\{\,\} \sqsubseteq \not\downarrow\mathsf{E}\{\,\}} \qquad \frac{\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}\{\,\}}{\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}\{\,\}\not\searrow} \qquad \frac{\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}_1\{\,\} \quad \mathsf{C}\{\,\} \sqsubseteq \mathsf{E}_2\{\,\}}{\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}_1\{\,\} \wedge \mathsf{E}_2\{\,\}}$$

(→) reduction *The (→) reduction rule is the union of the three reduction rules defines as follows:*

if $\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}\{\,\}$ *then*

$$\mathsf{C}\{(\lambda x{:}\iota.M)\, N\} \mathbin{@} \mathsf{E}\{(\lambda \iota{:}\sigma_i.\Delta_i)\, \Delta'_i\}_{i\in I} \to_\beta \mathsf{C}\{M[N/x]\} \mathbin{@} \mathsf{E}\{\Delta_i[\Delta'_i/\iota]\}_{i\in I}$$

$$M \mathbin{@} \mathsf{D}\{\not\downarrow(\Delta_1 \wedge \Delta_2)\} \to_{\pi_1} M \mathbin{@} \mathsf{D}\{\Delta_1\}$$

$$M \mathbin{@} \mathsf{D}\{(\Delta_1 \wedge \Delta_2)\not\searrow\} \to_{\pi_2} M \mathbin{@} \mathsf{D}\{\Delta_2\}$$

Note that the hypothesis of $\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}\{\,\}$ in the definition of the $\to_\beta$-reduction is essential in order to recover, for each β-redex in the marked term, the corresponding β-redexes, wrapped in the proof-term; those redexes must fired in parallel, in order to do not create exotic typed terms that would not have an untyped counterpart in the original system *à la* Curry.

α-conversion on well formed terms can be formally defined as follows:

Definition 3.4 (α-conversion)

$$(\lambda x{:}\iota.M) \mathbin{@} \Delta \to_\alpha (\lambda y{:}\iota.M[y/x]) \mathbin{@} \Delta \qquad y \textit{ fresh in } M$$

$$M \mathbin{@} (\lambda \iota_1{:}\sigma.\Delta) \to_\alpha M[\iota_2/\iota_1] \mathbin{@} (\lambda \iota_2{:}\sigma.\Delta[\iota_2/\iota_1]) \qquad \iota_2 \textit{ fresh in } \Delta$$

4 Examples

We show two notorious examples in order to justify how type derivations can be built for $\Lambda^{\mathsf{t}}_\wedge$ proof-terms starting for the corresponding untyped λ-terms *à la* Curry, and two further examples in order to illustrate the reduction rules of $\Lambda^{\mathsf{t}}_\wedge$.

Example 4.1 (Classical polymorphic identity)
We show a derivation for a typed term corresponding to the typing in $\Lambda^{\mathsf{u}}_\wedge$*:* $\vdash_\wedge \lambda x.x : (\sigma_1 \to \sigma_1) \wedge (\sigma_2 \to \sigma_2)$.

$$\dfrac{\dfrac{\dfrac{}{x@0{:}\sigma_1 \vdash x@0 : \sigma_1}(Var)}{\vdash (\lambda x{:}0.x)@(\lambda 0{:}\sigma_1.0) : \sigma_1 \to \sigma_1}(\to I) \quad \dfrac{\dfrac{}{x@0{:}\sigma_2 \vdash x@0 : \sigma_2}(Var)}{\vdash (\lambda x{:}0.x)@(\lambda 0{:}\sigma_2.0) : \sigma_2 \to \sigma_2}(\to I)}{\vdash (\lambda x{:}0.x)@((\lambda 0{:}\sigma_1.0) \wedge (\lambda 0{:}\sigma_2.0)) : (\sigma_1 \to \sigma_1) \wedge (\sigma_2 \to \sigma_2)}(\wedge I)$$

Example 4.2 (Polymorphic self-application)
Let $\sigma_2 \triangleq (\sigma_1 \to \sigma_1) \wedge \sigma_1$. We show a term in $\Lambda^{\mathsf{t}}_{\wedge}$ corresponding to the typing of $\Lambda^{\mathsf{u}}_{\wedge}$: $\vdash_{\wedge} \lambda x.xx : \sigma_2 \to \sigma_1$.

$$\dfrac{\dfrac{\dfrac{}{x@0{:}\sigma_2 \vdash x@0 : \sigma_2}(Var)}{x@0{:}\sigma_2 \vdash x@(\swarrow 0) : \sigma_1 \to \sigma_1}(\wedge E_L) \quad \dfrac{\dfrac{}{x{:}0{:}\sigma_2 \vdash x@0 : \sigma_2}(Var)}{x{:}0{:}\sigma_2 \vdash x@(0\searrow) : \sigma_1}(\wedge E_R)}{x@0{:}\sigma_2 \vdash (x\,x)@(\swarrow 0)\,(0\searrow) : \sigma_1}(\to E)}{\vdash (\lambda x{:}0.x\,x)@(\lambda 0{:}\sigma_2.(\swarrow 0)\,(0\searrow)) : \sigma_2 \to \sigma_1}(\to I)$$

Note how the proof-tree memorizes exactly the skeleton of the type-derivation.

Example 4.3 (Reduction in $\Lambda^{\mathsf{t}}_{\wedge}$)
A term in $\Lambda^{\mathsf{t}}_{\wedge}$ corresponding to the following typing in $\Lambda^{\mathsf{u}}_{\wedge}$: $y{:}\sigma_1 \wedge \sigma_2 \vdash_{\wedge} (\lambda x.x)\,y : \sigma_1 \wedge \sigma_2$ can be constructed in the following way:

$$\dfrac{\mathcal{D}_1 \qquad\qquad \mathcal{D}_2}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash (\lambda x{:}1.x)\,y@((\lambda 1{:}\sigma_1.1)\swarrow 0) \wedge ((\lambda 1{:}\sigma_2.1)\,0\searrow) : \sigma_1 \wedge \sigma_2}(\wedge I)$$

where $\mathcal{D}_1$ is:

$$\dfrac{\dfrac{\dfrac{}{y@0{:}\sigma_1 \wedge \sigma_2, x@1{:}\sigma_1 \vdash x@1 : \sigma_1}(Var)}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash (\lambda x{:}1.x)@(\lambda 1{:}\sigma_1.1) : \sigma_1 \to \sigma_1}(\to I) \quad \dfrac{\dfrac{}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash y@0 : \sigma_1 \wedge \sigma_2}(Var)}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash y@\swarrow 0 : \sigma_1}(\wedge E_L)}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash (\lambda x{:}1.x)\,y@((\lambda 1{:}\sigma_1.1)\swarrow 0) : \sigma_1}(\to E)$$

and $\mathcal{D}_2$ is:

$$\dfrac{\dfrac{\dfrac{}{y@0{:}\sigma_1 \wedge \sigma_2, x@1{:}\sigma_2 \vdash x@1 : \sigma_2}(Var)}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash (\lambda x{:}1.x)@(\lambda 1{:}\sigma_2.1) : \sigma_2 \to \sigma_2}(\to I) \quad \dfrac{\dfrac{}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash y@0 : \sigma_1 \wedge \sigma_2}(Var)}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash y@0\searrow : \sigma_2}(\wedge E_R)}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash (\lambda x{:}1.x)\,y@((\lambda 1{:}\sigma_2.1)\,0\searrow) : \sigma_2}(\to E)$$

This term can be written as: $\mathsf{C}\{(\lambda x{:}1.x)\,y\}@\mathsf{E}\{(\lambda 1{:}\sigma_1.1)\swarrow 0\}\{(\lambda 1{:}\sigma_2.1)\,0\searrow\}$, where the two overlapping contexts are respectively: $\mathsf{C}\{\,\} \equiv \{\,\}$ and $\mathsf{E}\{\,\} \equiv \{\,\} \wedge \{\,\}$, so this term β-reduces to: $y@\swarrow 0 \wedge 0\searrow$ which is a well formed term, in fact it can be built in the following way:

$$\dfrac{\dfrac{\dfrac{}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash y@0 : \sigma_1 \wedge \sigma_2}(Var)}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash y@\swarrow 0 : \sigma_1}(\wedge E_L) \quad \dfrac{\dfrac{}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash y@0 : \sigma_1 \wedge \sigma_2}(Var)}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash y@0\searrow : \sigma_2}(\wedge E_R)}{y@0{:}\sigma_1 \wedge \sigma_2 \vdash y@\swarrow 0 \wedge 0\searrow : \sigma_1 \wedge \sigma_2}(\wedge I)$$

Note that, if only one of the redexes in the proof-term would be reduced, then the resulting term will be not well formed.

Example 4.4 (A complete reduction)
Consider the typing in $\Lambda^{\mathsf{u}}_{\wedge}$

$$\vdash_{\wedge} (\lambda x.x\,x)\,((\lambda y.y)\,(\lambda y.y)) : \sigma \wedge \tau$$

A corresponding term in $\Lambda^{\mathsf{t}}_{\wedge}$ *is:*

$$\vdash (\lambda x{:}0.x\,x)\,((\lambda y{:}1.y)\,(\lambda y{:}2.y))@(\Delta_1\,\Delta_2) : \sigma \wedge \tau$$

Let

$$\delta \triangleq ((\sigma{\to}\sigma) \wedge \sigma) \wedge ((\tau{\to}\tau) \wedge \tau)$$

and

$$\begin{array}{llll}
\Delta_1 = \lambda 0 : \delta.\Delta_3 & \Delta_3 = \Delta_4 \wedge \Delta_5 & \Delta_4 = \Delta_6\,\Delta_7 & \Delta_5 = \Delta'_6\,\Delta'_7 \\
\Delta_6 = \swarrow(\swarrow 0) & \Delta_7 = (\swarrow 0)\searrow & \Delta'_6 = \swarrow(0\searrow) & \Delta'_7 = (0\searrow)\searrow \\
\Delta_2 = \Delta_8 \wedge \Delta_9 & \Delta_8 = \Delta_{10} \wedge \Delta_{11} & \Delta_{10} = \Delta_{12}\,\Delta_{13} & \Delta_{12} = \lambda 1{:}\sigma^2.2 \\
\Delta_{13} = \lambda 2{:}\sigma.2 & \Delta_{11} = \Delta_{14}\,\Delta_{15} & \Delta_{14} = \lambda 1{:}\sigma.1 & \Delta_{15} = \lambda 2{:}\alpha.2 \\
\Delta_9 = \Delta'_{10} \wedge \Delta'_{11} & \Delta'_{10} = \Delta'_{12}\,\Delta'_{13} & \Delta'_{12} = \lambda 1{:}\tau^2.1 & \Delta'_{13} = \lambda 2{:}\tau.2 \\
\Delta'_{11} = \Delta'_{14}\,\Delta'_{15} & \Delta'_{14} = \lambda 1{:}\tau.1 & \Delta'_{15} = \lambda 2{:}\beta.1 &
\end{array}$$

Reducing the top-level β*-redex (corresponding to the two overlapping contexts* $\mathsf{C}\{\,\} \equiv \mathsf{E}\{\,\} \equiv \{\,\}$*), leads to the term*

$$((\lambda y{:}1.y)\,(\lambda y{:}2.y))\,((\lambda y{:}1.y)\,(\lambda y{:}2.y))@((\underbrace{\swarrow(\swarrow\Delta_2)}_{\pi-red})\,(\underbrace{(\swarrow\Delta_2)\searrow}_{\pi-red}))\wedge((\underbrace{\swarrow(\Delta_2\searrow)}_{\pi-red})\,(\underbrace{(\Delta_2\searrow)\searrow}_{\pi-red}))$$

Reducing all the four π*-redexes leads to the following term*

$$((\lambda y{:}1.y)\,(\lambda y{:}2.y))\,((\lambda y{:}1.y)\,(\lambda y{:}2.y))@(\Delta_{10}\,\Delta_{11}) \wedge (\Delta'_{10}\,\Delta'_{11})$$

that can be seen both as:

$$\mathsf{C}\{(\lambda y{:}1.y)\,(\lambda y{:}2.y)\}@\mathsf{E}\{\Delta_{10}\}\{\Delta'_{10}\}$$

and

$$\mathsf{C}\{(\lambda y{:}1.y)\,(\lambda y{:}2.y)\}@\mathsf{E}\{\Delta_{11}\}\{\Delta'_{11}\}$$

by considering either the contexts $\mathsf{C}\{\,\} \equiv \{\,\}((\lambda y{:}1.y)\,(\lambda y{:}2.y))$ *and* $\mathsf{E}\{\,\} \equiv \{\,\}\,\Delta_{11} \wedge \{\,\}\,\Delta'_{11}$ *or* $\mathsf{C}\{\,\} \equiv ((\lambda y{:}1.y)\,(\lambda y{:}2.y))\{\,\}$ *and* $\mathsf{E}\{\,\} \equiv \Delta_{10}\,\{\,\} \wedge \Delta'_{10}\,\{\,\}$.

By making the first choice, we obtain:

$$(\lambda y{:}2.y)\,((\lambda y{:}1.y)\,(\lambda y{:}2.y))@(\Delta_{13}\,\Delta_{11}) \wedge (\Delta'_{13}\,\Delta'_{11})$$

where the two overlapping contexts are: $\mathsf{C}\{\,\} \equiv (\lambda y{:}2.y)\,\{\,\}$ *and* $\mathsf{E}\{\,\} \equiv (\Delta_{13}\,\{\,\}) \wedge (\Delta'_{13}\,\{\,\})$.

Reducing this β*-redex leads to the following term*

$$(\lambda y{:}2.y)\,(\lambda y{:}2.y)@(\Delta_{13}\,\Delta_{15}) \wedge (\Delta'_{13}\,\Delta'_{15})$$

where the overlapping contexts $\mathsf{C}\{\,\} \equiv \{\,\}$ *and* $\mathsf{E}\{\,\} \equiv \{\,\} \wedge \{\,\}$ *individuate the last* β*-redex. Reducing this* β*-redex we obtain the term in normal form*

$$(\lambda y{:}2.y)@\Delta_{15} \wedge \Delta'_{15}$$

5 The Isomorphism between $\Lambda^{\mathsf{u}}_{\wedge}$ and $\Lambda^{\mathsf{t}}_{\wedge}$

In this section we prove that the type system $\vdash$ for $\Lambda^{\mathsf{t}}_{\wedge}$ is isomorphic to the classical system $\vdash_{\wedge}$ for $\Lambda^{\mathsf{u}}_{\wedge}$ of Coppo and Dezani [Coppo and Dezani-Ciancaglini (1980)]. The isomorphism is given for a customization of the general definition of isomorphism given in [Giannini et al. (1993), Liquori (1996), van Bakel et al. (1997)], to the case of intersection-types and proof-trees.

From the logical point of view, the existence of an isomorphism means that there is a one-to-one correspondence between the judgments that can be proved in the two systems, and the derivations correspond with each other rule by rule. In what follows, and with a little abuse of notation, marked-terms and untyped terms of the λ-calculus will be ranged over by $M, N, \ldots$, the difference between marked-terms and untyped-terms being clear from the context (*i.e.* the judgment to be proved).

Definition 5.1 (Church *vs.* Curry)

(1) The type-erasing function $\mathcal{E} : \Lambda^{\mathsf{t}}_{\wedge} \Rightarrow \Lambda$ *is inductively defined on terms as follows:*

$$\mathcal{E}(x@\_) \triangleq x$$

$$\mathcal{E}((\lambda x{:}\iota.M)@\_) \triangleq \lambda x.\mathcal{E}(M@\_)$$

$$\mathcal{E}((M\,N)@\_) \triangleq \mathcal{E}(M@\_)\,\mathcal{E}(N@\_)$$

$\mathcal{E}$ *can be extended to contexts in the following way:*

$$\mathcal{E}(\epsilon) \triangleq \epsilon$$

$$\mathcal{E}(\Gamma, x@\iota{:}\sigma) \triangleq \mathcal{E}(\Gamma), x{:}\sigma$$

(2) Let $\mathcal{D}er\Lambda^{\mathsf{u}}_{\wedge}$ *and* $\mathcal{D}er\Lambda^{\mathsf{t}}_{\wedge}$ *be the sets of all (un)typed derivations in* $\vdash_{\wedge}$ *and* $\vdash$*, respectively. Let* $\mathcal{D}, \mathcal{D}_1, \ldots, \mathcal{D}_n$ *range over (un)typed derivations. Systems* $\vdash_{\wedge}$

$$\mathcal{G}\left(\frac{x{:}\sigma \in E}{E \vdash_\wedge x : \sigma}(Var)\right) \triangleq \begin{cases} \dfrac{x@\iota{:}\sigma \in \Gamma}{\Gamma \vdash x@\iota : \sigma}(Var) \\ \mathcal{E}(\Gamma) = E \quad \iota \text{ is fresh} \end{cases}$$

$$\mathcal{G}\left(\frac{\mathcal{D} : E, x{:}\sigma_1 \vdash_\wedge M' : \sigma_2}{E \vdash_\wedge \lambda x.M' : \sigma_1 \to \sigma_2}(\to I)\right) \triangleq \begin{cases} \dfrac{\mathcal{G}(\mathcal{D}) : \Gamma, x@\iota{:}\sigma_1 \vdash M@\Delta : \sigma_2}{\Gamma \vdash (\lambda x{:}\iota.M)@(\lambda\iota{:}\sigma_1.\Delta) : \sigma_1 \to \sigma_2}(\to I) \\ \mathcal{E}(\Gamma, x@\iota{:}\sigma_1) = E, x{:}\sigma_1 \;\&\; \mathcal{E}(M@\Delta) = M' \end{cases}$$

$$\mathcal{G}\left(\frac{\mathcal{D}_1 : E \vdash_\wedge M' : \sigma_1 \to \sigma_2 \quad \mathcal{D}_2 : E \vdash_\wedge N : \sigma_1}{E \vdash_\wedge M'\,N' : \sigma_2}(\to E)\right) \triangleq \begin{cases} \dfrac{\mathcal{G}(\mathcal{D}_1) : \Gamma \vdash M@\Delta_1 : \sigma_1 \to \sigma_2 \quad \mathcal{G}(\mathcal{D}_2) : \Gamma \vdash N@\Delta_2 : \sigma_1}{\Gamma \vdash (M\,N)@(\Delta_1\,\Delta_2) : \sigma_2}(\to E) \\ \mathcal{E}(\Gamma) = E \;\&\; \mathcal{E}(M@\Delta_1) = M' \;\&\; \mathcal{E}(N@\Delta_2) = N' \end{cases}$$

$$\mathcal{G}\left(\frac{\mathcal{D}_1 : E \vdash_\wedge M' : \sigma_1 \quad \mathcal{D}_2 : E \vdash_\wedge M' : \sigma_2}{E \vdash_\wedge M' : \sigma_1 \wedge \sigma_2}(\wedge I)\right) \triangleq \begin{cases} \dfrac{\mathcal{G}(\mathcal{D}_1) : \Gamma \vdash M@\Delta_1 : \sigma_1 \quad \mathcal{G}(\mathcal{D}_2) : \Gamma \vdash M@\Delta_2 : \sigma_2}{\Gamma \vdash M@(\Delta_1 \wedge \Delta_2) : \sigma_1 \wedge \sigma_2}(\wedge I) \\ \mathcal{E}(\Gamma) = E \;\&\; \mathcal{E}(M@(\Delta_1 \wedge \Delta_2)) = M' \end{cases}$$

$$\mathcal{G}\left(\frac{\mathcal{D} : E \vdash_\wedge M' : \sigma_1 \wedge \sigma_2}{E \vdash_\wedge M' : \sigma_1}(\wedge E_L)\right) \triangleq \begin{cases} \dfrac{\mathcal{G}(\mathcal{D}) : \Gamma \vdash M@\Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\swarrow\Delta) : \sigma_1}(\wedge E_L) \\ \mathcal{E}(\Gamma) = E \;\&\; \mathcal{E}(M@\Delta) = M' \end{cases}$$

$$\mathcal{G}\left(\frac{\mathcal{D} : E \vdash_\wedge M' : \sigma_1 \wedge \sigma_2}{E \vdash_\wedge M' : \sigma_2}(\wedge E_R)\right) \triangleq \begin{cases} \dfrac{\mathcal{G}(\mathcal{D}) : \Gamma \vdash M@\Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\Delta\searrow) : \sigma_2}(\wedge E_R) \\ \mathcal{E}(\Gamma) = E \;\&\; \mathcal{E}(M@\Delta) = M' \end{cases}$$

Figure 5. The Function $\mathcal{G}$.

and $\vdash$ are isomorphic, if and only if there exist $\mathcal{F} : \mathcal{D}er\Lambda^{\mathsf{t}}_\wedge \Rightarrow \mathcal{D}er\Lambda^{\mathsf{u}}_\wedge$ and $\mathcal{G} : \mathcal{D}er\Lambda^{\mathsf{u}}_\wedge \Rightarrow \mathcal{D}er\Lambda^{\mathsf{t}}_\wedge$, such that:

(a) (Soundness) If $\mathcal{D} : \Gamma \vdash M@\Delta : \sigma$, then $\mathcal{F}(\mathcal{D}) : \mathcal{E}(\Gamma) \vdash_\wedge \mathcal{E}(M@\Delta) : \sigma$;

(b) (Completeness) If $\mathcal{D} : E \vdash_\wedge M' : \sigma$, then there exists Γ and Δ, such that $\mathcal{G}(\mathcal{D}) : \Gamma \vdash M@\Delta : \sigma$, and $\mathcal{E}(\Gamma) \equiv E$, with $\mathcal{E}(M@\Delta) \equiv M'$;

(c) (Inversion) $\mathcal{F} \circ \mathcal{G}$ is the identity in $\mathcal{D}er\Lambda^{\mathsf{u}}_\wedge$, and $\mathcal{G} \circ \mathcal{F}$ is the identity in $\mathcal{D}er\Lambda^{\mathsf{t}}_\wedge$, modulo uniform naming of variable-marks. I.e.,

$$\mathcal{G}(\mathcal{F}(\Gamma \vdash M@\Delta : \sigma)) = \mathsf{ren}(\Gamma) \vdash \mathsf{ren}(M@\Delta) : \sigma$$

where ren *is a is a simple function renaming the free occurrences of variable-marks;*

(d) (Faithfulness) Both $\mathcal{F}$ and $\mathcal{G}$ preserve the structure of derivations, (i.e., the tree obtained from a derivation by erasing all judgments, but not the names of the rules).

$$\mathcal{F}\left(\frac{x@\iota{:}\sigma \in \Gamma}{\Gamma \vdash x@\iota : \sigma}\,(Var)\right) \triangleq \begin{cases} \dfrac{x{:}\sigma \in E}{E \vdash_\wedge x : \sigma}\,(Var) \\ \mathcal{E}(\Gamma) = E \end{cases}$$

$$\mathcal{F}\left(\frac{\Gamma, x@\iota{:}\sigma_1 \vdash M@\Delta : \sigma_2 \quad (\to I)}{\Gamma \vdash (\lambda x{:}\iota.M)@(\lambda\iota{:}\sigma_1.\Delta) : \sigma_1 \to \sigma_2}\right) \triangleq \begin{cases} \dfrac{\mathcal{F}(\mathcal{D}) : E, x{:}\sigma_1 \vdash_\wedge M' : \sigma_2}{E \vdash_\wedge \lambda x.M' : \sigma_1 \to \sigma_2}\,(\to I) \\ \mathcal{E}(\Gamma, x@\iota{:}\sigma_1) = E, x{:}\sigma_1 \;\&\; \mathcal{E}(M@\Delta) = M' \end{cases}$$

$$\mathcal{F}\left(\frac{\begin{array}{l}\mathcal{D}_1 : \Gamma \vdash M@\Delta_1 : \sigma_1 \to \sigma_2 \\ \mathcal{D}_2 : \Gamma \vdash N@\Delta_2 : \sigma_1\end{array}}{\Gamma \vdash (M\,N)@(\Delta_1\,\Delta_2) : \sigma_2}\,(\to E)\right) \triangleq \begin{cases} \dfrac{\begin{array}{l}\mathcal{F}(\mathcal{D}_1) : E \vdash_\wedge M' : \sigma_1 \to \sigma_2 \\ \mathcal{F}(\mathcal{D}_2) : E \vdash_\wedge N' : \sigma_1\end{array}}{E \vdash_\wedge M'\,N' : \sigma_2}\,(\to E) \\ \mathcal{E}(\Gamma) = E \;\&\; \mathcal{E}(M@\Delta_1) = M' \;\&\; \mathcal{E}(N@\Delta_2) = N' \end{cases}$$

$$\mathcal{F}\left(\frac{\begin{array}{c}\mathcal{D}_1 : \Gamma \vdash M@\Delta_1 : \sigma_1 \\ \mathcal{D}_2 : \Gamma \vdash M@\Delta_2 : \sigma_2\end{array}}{\Gamma \vdash M@(\Delta_1 \wedge \Delta_2) : \sigma_1 \wedge \sigma_2}\,(\wedge I)\right) \triangleq \begin{cases} \dfrac{\begin{array}{l}\mathcal{F}(\mathcal{D}_1) : E \vdash_\wedge M' : \sigma_1 \\ \mathcal{F}(\mathcal{D}_2) : E \vdash_\wedge M' : \sigma_2\end{array}}{E \vdash_\wedge M' : \sigma_1 \wedge \sigma_2}\,(\wedge I) \\ \mathcal{E}(\Gamma) = E \;\&\; \mathcal{E}(M@(\Delta_1 \wedge \Delta_2)) = M' \end{cases}$$

$$\mathcal{F}\left(\frac{\mathcal{D} : \Gamma \vdash M@\Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\swarrow\Delta) : \sigma_1}\,(\wedge E_L)\right) \triangleq \begin{cases} \dfrac{\mathcal{F}(\mathcal{D}) : E \vdash_\wedge M' : \sigma_1 \wedge \sigma_2}{E \vdash_\wedge M' : \sigma_1}\,(\wedge E_L) \\ \mathcal{E}(\Gamma) = E \;\&\; \mathcal{E}(M@\Delta) = M' \end{cases}$$

$$\mathcal{F}\left(\frac{\mathcal{D} : \Gamma \vdash M@\Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\Delta\searrow) : \sigma_2}\,(\wedge E_R)\right) \triangleq \begin{cases} \dfrac{\mathcal{F}(\mathcal{D}) : E \vdash_\wedge M' : \sigma_1 \wedge \sigma_2}{E \vdash_\wedge M' : \sigma_2}\,(\wedge E_R) \\ \mathcal{E}(\Gamma) = E \;\&\; \mathcal{E}(M@\Delta) = M' \end{cases}$$

Figure 6. The Function $\mathcal{F}$.

Function $\mathcal{F}$ and $\mathcal{G}$ are described in Figures 6 and 5.

Notice that the definition of isomorphism expresses more than just soundness and completeness of $\mathcal{E}$. Indeed, soundness and completeness imply an isomorphism between the judgments of the two systems, but they do not imply necessarily a one-one correspondence between derivations. Figure 7 shows the various functions between typed and untyped systems of λ-calculi that realize the above relations between typed and untyped judgments and derivations.

Theorem 5.1 (Isomorphism) *The systems $\vdash$ and $\vdash_\wedge$ are isomorphic.*

Proof. Soundness can be proved by induction on the structure of the derivation in the $\Lambda^{\mathsf{t}}_\wedge$. Completeness can be proved by induction on the structure of the derivation in $\Lambda^{\mathsf{u}}_\wedge$, using soundness. Inversion can be proved by induction on the structure of both the derivations, using the soundness and completeness result. Faithfulness is

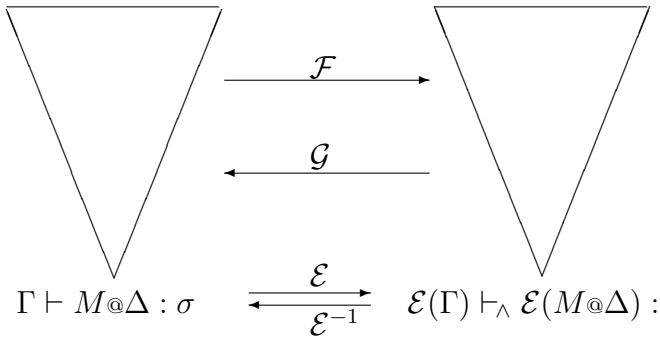

Figure 7. Functions between $\Lambda^{\mathsf{t}}_\wedge$s and $\Lambda^{\mathsf{u}}_\wedge$s Judgments and Derivations.

immediate. □

We can also explore the relationship between $\Lambda^{\mathsf{t}}_\wedge$ and the proof calculus $\Lambda\mathcal{P}$ by defining an erasure function $\mathcal{E}' : \Lambda^{\mathsf{t}}_\wedge \Rightarrow \Lambda\mathcal{P}$ as follows:

$$\mathcal{E}'(\_@\Delta) \triangleq \Delta$$

The function $\mathcal{E}'$ can be extended naturally to a function from contexts to proof-contexts:

$$\mathcal{E}'(\epsilon) \triangleq \epsilon$$

$$\mathcal{E}'(\Gamma, x@\iota{:}\sigma) \triangleq \mathcal{E}'(\Gamma), \iota{:}\sigma$$

Then it is easy to define a function $\mathcal{F}' : \mathcal{D}er\Lambda^{\mathsf{t}}_\wedge \Rightarrow \mathcal{D}er\Lambda\mathcal{P}$ such that $\mathcal{D} : \Gamma \vdash M@\Delta : \sigma$ implies $\mathcal{F}'(\mathcal{D}) : \mathcal{E}'(\Gamma) \vdash_P \Delta : \sigma$. $\mathcal{F}'$ consists in just applying $\mathcal{E}'$ to all contexts and subjects of the derivation. Note that $\Lambda^{\mathsf{t}}_\wedge$ and $\Lambda\mathcal{P}$ are not isomorphic; for example, the statement $\iota_1{:}\sigma \vdash_P (\lambda\iota_2{:}\tau.\iota_2) \wedge \iota_1 : (\tau{\rightarrow}\tau) \wedge \sigma$ in $\Lambda\mathcal{P}$ has no a corresponding counterpart in $\Lambda^{\mathsf{t}}_\wedge$.

6 Metatheory of $\Lambda^{\mathsf{t}}_\wedge$

In this section we will prove properties [3-6] listed in the introduction. First we need to prove the Generation and Substitution lemmas.

Lemma 6.1 (Generation)

(1) If $\Gamma \vdash M@\Delta_1 \wedge \Delta_2 : \sigma$*, then* $\sigma \equiv \sigma_1 \wedge \sigma_2$ *and* $\Gamma \vdash M@\Delta_1 : \sigma_1$ *and* $\Gamma \vdash M@\Delta_2 : \sigma_2$*;*
(2) If $\Gamma \vdash M@\not\downarrow\Delta : \sigma$ *then there exists* τ*, such that* $\Gamma \vdash M@\Delta : \sigma \wedge \tau$*;*
(3) If $\Gamma \vdash M@\Delta\!\searrow : \sigma$ *then there exists* τ*, such that* $\Gamma \vdash M@\Delta : \tau \wedge \sigma$*;*
(4) If $\Gamma \vdash x@\iota : \sigma$*, then* $x@\iota : \sigma \in \Gamma$*;*
(5) If $\Gamma \vdash (\lambda x{:}\iota.M)@(\lambda\iota{:}\sigma_1.\Delta) : \sigma_1{\rightarrow}\sigma_2$*, then* $\Gamma, x@\iota{:}\sigma_1 \vdash M@\Delta : \sigma_2$*;*

(6) If $\Gamma \vdash M\,N@\Delta_1\,\Delta_2 : \sigma$*, then there exists* σ'*, such that* $\Gamma \vdash M@\Delta_1 : \sigma'{\to}\sigma$*, and* $\Gamma \vdash N@\Delta_2 : \sigma'$*.*

Proof. By induction on derivation. □

Lemma 6.2 (Substitution)
If $\Gamma, x@\iota{:}\sigma \vdash M@\Delta : \tau$*, and* $\Gamma \vdash N@\Delta' : \sigma$*, then* $\Gamma \vdash M[N/x]@\Delta[\Delta'/\iota] : \tau$*.*

Proof. By induction on derivation. □

We continue with subject reduction

Theorem 6.1 (Subject reduction)
If $\Gamma \vdash M@\Delta : \sigma$ *and* $M@\Delta \to N@\Delta'$*, then* $\Gamma \vdash N@\Delta' : \sigma$*.*

Proof. We will treat the case $\to$ is $\to_\beta$; the cases $\to_{\pi_1}$ and $\to_{\pi_2}$ are easier. By definition, there are two contexts $\mathsf{C}\{\,\}$ and $\mathsf{E}\{\,\}$, such that $\mathsf{C}\{\,\} \sqsubseteq \mathsf{E}\{\,\}$, and $M@\Delta \equiv \mathsf{C}\{(\lambda x{:}\iota.P)\,Q\}@\mathsf{E}\{(\lambda\iota{:}\sigma_i.\Delta_i)\,\Delta'_i\}_{i\in I}$ which β-reduces to $\mathsf{C}\{P[Q/x]\}@\mathsf{E}\{\Delta_i[\Delta'_i/\iota]\}_{i\in I}$.

By induction on $\mathsf{C}\{\,\}$. Let $\mathsf{C}\{\,\} \equiv \{\,\}$. We proceed now by induction on $\mathsf{E}\{\,\}$. If $\mathsf{E}\{\,\} \equiv \{\,\}$, then, by Generation Lemma.$(5-6)$ the derivation is of the shape:

$$\dfrac{\dfrac{\Gamma, x@\iota{:}\tau \vdash P@\Delta : \sigma}{\Gamma \vdash \lambda x{:}\iota.P@\lambda\iota{:}\tau.\Delta : \tau{\to}\sigma}(\to I) \qquad \Gamma \vdash Q@\Delta' : \tau}{\Gamma \vdash (\lambda x{:}\iota.P)\,Q@(\lambda\iota{:}\tau.\Delta)\,\Delta' : \sigma}(\to E)$$

Then, by Substitution Lemma, we get $\Gamma \vdash P[Q/x]@\Delta[\Delta'/\iota] : \sigma$.

Let $\mathsf{E}\{\,\} \equiv \mathsf{E}_1\{\,\} \wedge \mathsf{E}_2\{\,\}$, where $\{\,\} \sqsubseteq \mathsf{E}_i\{\,\}$, and $1 \le i \le 2$. Then, by Generation Lemma.(1), we have $\sigma \equiv \sigma_1 \wedge \sigma_2$ and $\Gamma \vdash (\lambda x{:}\iota.P)\,Q@\mathsf{E}_1\{(\lambda\iota{:}\sigma_i.\Delta_i)\,\Delta'_i\}_{i\in I_1} : \sigma_1$ and $\Gamma \vdash (\lambda x{:}\iota.P)\,Q@\mathsf{E}_2\{(\lambda\iota{:}\sigma_i.\Delta_i)\,\Delta'_i\}_{i\in I_2} : \sigma_2$, for some I_1, I_2, such that $I = I_1 \cup I_2$. By induction, we have $\Gamma \vdash P[Q/x]@\mathsf{E}_1\{\Delta_i[\Delta'_i/\iota]\}_{i\in I_1}$, and $\Gamma \vdash P[Q/x]@\mathsf{E}_2\{\Delta_i[\Delta'_i/\iota]\}_{i\in I_2}$, and the proof follows, by rule $(\wedge I)$.

The cases for $\mathsf{E}\{\,\} \equiv \not\downarrow\mathsf{E}_1\{\,\}$, or $\mathsf{E}\{\,\} \equiv \mathsf{E}_1\{\,\}\!\downarrow$ follow easily by induction. No other cases can apply.

For the inductive case, we will show the proof in case $\mathsf{C}\{\,\} \equiv R\,\mathsf{C}_1\{\,\}$, and $\mathsf{E}\{\,\} \equiv \Delta\,\mathsf{E}_1\{\,\}$, with $\mathsf{C}_1\{\,\} \sqsubseteq \mathsf{E}_1\{\,\}$. Then $R\,\mathsf{C}_1\{(\lambda x{:}\iota.P)\,Q\}@\Delta\,\mathsf{E}_1\{(\lambda\iota{:}\sigma_i.\Delta_i)\,\Delta'_i\}_{i\in I} \to_\beta R\,\mathsf{C}_1\{P[Q/x]\}@\Delta\mathsf{E}_1\{\Delta_i[\Delta'_i/\iota]\}_{i\in I}$. By the Generation Lemma, there exists σ', such that $\Gamma \vdash R@\Delta : \sigma'{\to}\sigma$, and $\Gamma \vdash \mathsf{C}_1\{(\lambda x{:}\iota.P)\,Q\}@\mathsf{E}_1\{(\lambda\iota{:}\sigma_i.\Delta_i)\,\Delta'_i\}_{i\in I} : \sigma'$. By induction we get $\Gamma \vdash \mathsf{C}_1\{P[Q/x]\}@\mathsf{E}_1\{\Delta_i[\Delta'_i/\iota]\}_{i\in I} : \sigma'$, and the proof follows by rule $(\to E)$. The other cases are easy. □

$$
\begin{array}{lll}
\mathsf{Type}_{\wedge}(\Gamma, M@\Delta) & \stackrel{\triangle}{=} & \mathsf{match}\ (M@\Delta)\ \mathsf{with} \\
(\_@(\not\swarrow\Delta_1)) & \Rightarrow^1\ \sigma_1 & \mathsf{if}\ \mathsf{Type}_{\wedge}(\Gamma, M@\Delta_1) = \sigma_1 \wedge \sigma_2 \\
(\_@(\Delta_1\not\searrow)) & \Rightarrow^2\ \sigma_2 & \mathsf{if}\ \mathsf{Type}_{\wedge}(\Gamma, M@\Delta_1) = \sigma_1 \wedge \sigma_2 \\
(\_@(\Delta_1 \wedge \Delta_2)) & \Rightarrow^3\ \sigma_1 \wedge \sigma_2 & \mathsf{if}\ \mathsf{Type}_{\wedge}(\Gamma, M@\Delta_1) = \sigma_1\ \mathsf{and} \\
 & & \quad \mathsf{Type}_{\wedge}(\Gamma, M@\Delta_2) = \sigma_2 \\
(x@\_) & \Rightarrow^4\ \sigma & \mathsf{if}\ x@\iota{:}\sigma \in \Gamma \\
((\lambda x{:}\iota.M_1)@(\lambda\iota{:}\sigma_1.\Delta_1)) & \Rightarrow^5\ \sigma_1{\rightarrow}\sigma_2 & \mathsf{if}\ \mathsf{Type}_{\wedge}((\Gamma, x@\iota{:}\sigma_1), M_1@\Delta_1) = \sigma_2 \\
((M_1\ M_2)@(\Delta_1\ \Delta_2)) & \Rightarrow^6\ \sigma_2 & \mathsf{if}\ \mathsf{Type}_{\wedge}(\Gamma, M_1@\Delta_1) = \sigma_1{\rightarrow}\sigma_2\ \mathsf{and} \\
 & & \quad \mathsf{Type}_{\wedge}(\Gamma, M_2@\Delta_1) = \sigma_1 \\
(\_@\_) & \Rightarrow^7\ \mathsf{false} & \mathsf{otherwise}
\end{array}
$$

$$
\mathsf{Typecheck}_{\wedge}(\Gamma, M@\Delta, \sigma) \stackrel{\triangle}{=} \mathsf{Type}_{\wedge}(\Gamma, M@\Delta) \stackrel{?}{=} \sigma
$$

Figure 8. The Type Reconstruction and Type Checking Algorithms for $\Lambda^{\mathsf{t}}_{\wedge}$.

The strong normalization of $\Lambda^{\mathsf{t}}_{\wedge}$ is proved from the strong normalization of $\Lambda\mathcal{P}$.

Theorem 6.2 (Strong Normalization of $\Lambda^{\mathsf{t}}_{\wedge}$)
$\Lambda^{\mathsf{t}}_{\wedge}$ is strongly normalizing.

Proof. Let $\mathcal{D} : \Gamma \vdash M@\Delta : \sigma$, and let us assume, by *absurdum*, that there is an infinite reduction sequence starting from $M@\Delta$, *i.e.*,

$$
\begin{array}{l}
M@\Delta \equiv \mathsf{C}_1\{M_1\}@\mathsf{A}_1\{\Delta_1^i\}_{i\in I_1} \rightarrow \mathsf{C}_2\{M_2\}@\mathsf{A}_2\{\Delta_2^i\}_{i\in I_2} \rightarrow \ldots \rightarrow \\
\qquad \mathsf{C}_j\{M_j\}@\mathsf{A}_j\{\Delta_j^i\}_{i\in I_j} \rightarrow \mathsf{C}_{j+1}\{M_{j+1}\}@\mathsf{A}_{j+1}\{\Delta_{j+1}^i\}_{i\in I_{j+1}} \rightarrow \ldots
\end{array}
$$

where $\mathsf{A}\{\ \}$ is a context either of the shape $\mathsf{C}\{\ \}$, or of the shape $\mathsf{E}\{\ \}$. In the first case the reduction is a β-reduction, in the second one is a π-reduction. By applying the function $\mathcal{F}'$, defined at the end of Section 5, on each element of the reduction sequence, we get an infinite sequence of proof-terms $\mathsf{A}_1\{\Delta_1^i\}_{i\in I_1}, \mathsf{A}_2\{\Delta_2^i\}_{i\in I_2}$ $\ldots \mathsf{A}_j\{\Delta_j^i\}_{i\in I_j}, \mathsf{A}_{j+1}\{\Delta_{j+1}^i\}_{i\in I_{j+1}} \ldots$, such that, for all $j \geq 0$, we have that $\mathsf{A}_j\{\Delta_j^i\}_{i\in I_j}$ reduces to $\mathsf{A}_{j+1}\{\Delta_{j+1}^i\}_{i\in I_{j+1}}$, either by a π-reduction or by a strictly positive number of β-reductions. But this is impossible, since $\Lambda\mathcal{P}$ enjoys the strong normalization property (Fact 1). $\square$

The further requirement we asked for is the unicity of typing. In general, for typed languages, typing is unique modulo α-conversion, *i.e.*, modulo renaming of bound-variables.

Theorem 6.3 (Unicity of Typing of $\Lambda^{\mathsf{t}}_{\wedge}$)
If $\mathcal{D}_1 : \Gamma \vdash M@\Delta : \sigma_1$, *and* $\mathcal{D}_2 : \Gamma \vdash M'@\Delta' : \sigma_2$, *and* $M@\Delta =_\alpha M'@\Delta'$, *then* $\sigma_1 \equiv \sigma_2$, *and* $\mathcal{D}_1 \equiv \mathcal{D}_2$.

Proof. By easy induction on the structure of the derivation $\mathcal{D}_1$. □ We can finish this section by presenting the type reconstruction and the type checking algorithms for $\Lambda^{\mathsf{t}}_{\wedge}$ in Figure 6, and by proving that they are decidable. The soundness and completeness proofs follow.

Theorem 6.4 (Type Reconstruction for $\Lambda^{\mathsf{t}}_{\wedge}$)

(Soundness) *If* $\mathsf{Type}_\wedge(\Gamma, M@\Delta) = \sigma$, *then* $\Gamma \vdash M@\Delta : \sigma$
(Completeness) *If* $\Gamma \vdash M@\Delta : \sigma$, *then* $\mathsf{Type}_\wedge(\Gamma, M@\Delta) = \sigma$.

Proof.

(Soundness) By induction on the structure of $(M@\Delta)$.

- $(\_@(\not\downarrow\Delta_1))$ Then $\Delta \equiv \not\downarrow\Delta_1$ and $\sigma \equiv \sigma_1$. By induction, the judgment $\Gamma \vdash M@\Delta_1 : \sigma_1 \wedge \sigma_2$ is derivable. Apply rule $(\wedge E_L)$ to obtain a derivation for $\Gamma \vdash M@(\not\downarrow\Delta_1) : \sigma_1$.
- $(\_@(\Delta_1\not\downarrow))$ Then $\Delta \equiv \Delta_1\not\downarrow$ and $\sigma \equiv \sigma_2$. By induction, the judgment $\Gamma \vdash M@\Delta_1 : \sigma_1 \wedge \sigma_2$ is derivable. Apply rule $(\wedge E_R)$ to obtain a derivation for $\Gamma \vdash M@(\Delta_1\not\downarrow) : \sigma_2$.
- $(\_@(\Delta_1 \wedge \Delta_2))$ Then $\sigma \equiv \sigma_1 \wedge \sigma_2$. By induction, the judgments $\Gamma \vdash M@\Delta_1 : \sigma_1$ and $\Gamma \vdash M@\Delta_2 : \sigma_2$ are derivable. Apply rule $(\wedge I)$ to obtain a derivation for $\Gamma \vdash M@(\Delta_1 \wedge \Delta_2) : \sigma_1 \wedge \sigma_2$.
- $(x@\_)$ Then $M \equiv x$ and $\Delta \equiv \iota$, since the $\mathsf{Type}_\wedge$ algorithm (that works via a classical ML-like match-case analysis) has already ruled out the cases of $\Delta \in \{\not\downarrow\Delta_1, \Delta_1\not\downarrow, \Delta_1 \wedge \Delta_2\}$, and since the case $\Delta \equiv \lambda\iota{:}\sigma_1.\Delta_1$ does not apply. By hypothesis we get $x@\iota{:}\sigma \in \Gamma$. Apply rule (Var) to obtain a derivation for $\Gamma \vdash x@\iota : \sigma$.
- $((\lambda x{:}\iota.M_1)@(\lambda\iota{:}\sigma_1.\Delta_1))$ Then $M \equiv \lambda x{:}\iota.M_1$ and $\Delta \equiv \lambda\iota{:}\sigma_1.\Delta_1$ and $\sigma \equiv \sigma_1 \to \sigma_2$. By induction, the judgment $\Gamma, x@\iota{:}\sigma_1 \vdash M_1@\Delta_1 : \sigma_2$ is derivable. Apply rule $(\to I)$ to obtain a derivation for $\Gamma \vdash (\lambda x{:}\iota.M_1)@(\lambda\iota{:}\sigma_1.\Delta_1) : \sigma_1 \to \sigma_2$.
- $((M_1\, M_2)@(\Delta_1\, \Delta_2))$ Then $M \equiv M_1\, M_2$ and $\Delta \equiv \Delta_1\, \Delta_2$ and $\sigma \equiv \sigma_2$. By induction, the judgments $\Gamma \vdash M_1@\Delta_1 : \sigma_1 \to \sigma_2$ and $\Gamma \vdash M_2@\Delta_2 : \sigma_1$ are derivable. Apply rule $(\to E)$ to obtain a derivation for $\Gamma \vdash (M_1\, M_2)@(\Delta_1\, \Delta_2) : \sigma_2$.
- $(\_@\_)$ This case does not apply since $\sigma \neq \mathsf{false}$.

(Completeness) By induction on the derivation of $\Gamma \vdash M@\Delta : \sigma$.

- (Var) Then $M \equiv x$ and $\Delta \equiv \iota$. By match-case number 4 we get $\mathsf{Type}_\wedge(x, \iota@\sigma) = \sigma$.
- $(\rightarrow I)$ Then $M \equiv \lambda x{:}\iota.M_1$ and $\Delta \equiv \lambda\iota{:}\sigma_1.\Delta_1$ and $\sigma \equiv \sigma_1 \rightarrow \sigma_2$. By induction we get $\mathsf{Type}_\wedge((\Gamma, x@\iota{:}\sigma_1), M_1@\Delta_1) = \sigma_2$, and by match-case 5 we get $\mathsf{Type}_\wedge((\lambda x{:}\iota.M_1)@(\lambda\iota{:}\sigma_1.\Delta_1)) = \sigma_1 \rightarrow \sigma_2$.
- $(\rightarrow E)$ Then $M \equiv M_1\, M_2$ and $\Delta \equiv \Delta_1\, \Delta_2$ and $\sigma \equiv \sigma_2$. By induction we get $\mathsf{Type}_\wedge(\Gamma, M_1@\Delta_1) = \sigma_1 \rightarrow \sigma_2$ and $\mathsf{Type}_\wedge(\Gamma, M_2@\Delta_2) = \sigma_1$, and by match-case 6 we get $\mathsf{Type}_\wedge((M_1\, M_2)@(\Delta_1\, \Delta_2)) = \sigma_2$.
- $(\wedge I)$ Then $\Delta \equiv \Delta_1 \wedge \Delta_2$ and $\sigma \equiv \sigma_1 \wedge \sigma_2$. By induction we get $\mathsf{Type}_\wedge(M, \Delta_1) = \sigma_1$ and $\mathsf{Type}_\wedge(M, \Delta_2) = \sigma_2$, and by match-case 3 we get $\mathsf{Type}_\wedge(M, (\Delta_1 \wedge \Delta_2)) = \sigma_1 \wedge \sigma_2$.
- $(\wedge E_L)$ Then $\Delta \equiv \not\downarrow\Delta_1$ and $\sigma \equiv \sigma_1$. By induction we get $\mathsf{Type}_\wedge(\Gamma, M@\Delta_1) = \sigma_1 \wedge \sigma_2$, and by match-case 1 we get $\mathsf{Type}_\wedge(M, \not\downarrow\Delta_1) = \sigma_1$.
- $(\wedge E_R)$ Then $\Delta \equiv \Delta_1\!\downarrow$ and $\sigma \equiv \sigma_2$. By induction we get $\mathsf{Type}_\wedge(\Gamma, M@\Delta_1) = \sigma_1 \wedge \sigma_2$, and by match-case 2 we get $\mathsf{Type}_\wedge(M, \Delta_1\!\downarrow) = \sigma_2$.

□

Theorem 6.5 (Type Checking for $\Lambda^{\mathsf{t}}_\wedge$)
$\Gamma \vdash M@\Delta : \sigma$, *if and only if* $\mathsf{Typecheck}_\wedge(\Gamma, M@\Delta, \sigma) = \mathsf{true}$.

Proof. The $\Rightarrow$ part can be proved using completeness of the type reconstruction algorithm (Theorem 6.4), while the $\Leftarrow$ part can be proved using soundness of the type reconstruction algorithm. □

Theorem 6.6 (Judgment Decidability)
If is decidable whether the $\Lambda^{\mathsf{t}}_\wedge$ judgment $\Gamma \vdash M@\Delta : \sigma$ is derivable.

Proof. Routine. □

7 Conclusions

We studied in this paper the problem of designing a λ-calculus *à la* Church corresponding to the intersection-type assignment system. In particular, we asked for a typed language such that its relationship with the intersection-type assignment system enjoys all the standard requirements we posed in [Giannini et al. (1993), Liquori (1996), van Bakel et al. (1997)]. Examples of such "good" correspondences are respectively the Church and Curry version of the simple typed λ-calculus (if written using the same symbols), and the typed and type assignment version of the second order λ-calculus [Girard (1986), Leivant (1983)]. We succeed in designing a

calculus based essentially on two basic and simple ideas: an imperative-like notion of typing, when types are assigned to variables "at a given mark", and a proof-calculus, describing intersection-type derivations, whose terms are used as proof-trees for the terms of the target calculus.

A reader interested in particular in programming applications could object that the used language is far for being "usable", since the user needs to specify not only the typed terms, but also their proof-trees, which are encoding of type-derivations. The answer can be twofold. From a programming languages point of view, in every typed language the user, in order to write explicitly the type of a term, in some sense needs to "guess" the correct type-derivation assigning that type to the term itself. Here obviously the type-derivations are more difficult than in the simple typed case. But if we think, for example, to Girard's Second Order Typed λ-calculus [Girard (1986)], in order to write the term

$$\Lambda\beta.\Lambda\gamma.\lambda x{:}(\forall\alpha.\alpha).\, x\,(\beta{\to}\gamma) \;\text{ of type }\; \forall\beta.\forall\gamma.(\forall\alpha.\alpha){\to}(\beta{\to}\gamma)$$

one needs to know exactly how and the rules for introducing and eliminating the universal quantifier work. However, we think that the production of an usable language is not the only justification for the problem we studied, as it was especially for [Reynolds (1996), Pierce and Turner (1994)]. The relationship between typed and type assignment systems is an important theoretical issue, that is interesting in itself.

Acknowledgment. Simona was kindly supported by *QSL: Qualité et Sûrété du Logiciel, CPER, Région Lorraine*, Nancy, and by INRIA; Luigi was supported by the French CNRS grant *ACI Modulogic*. Moreover both authors want warmly thank the two anonymous referees, for their very sharp and constructive suggestions.

References

[Asperti A. et al. (2004)] Asperti A., Coppola P., Martini S. "(Optimal) duplication is not elementary recursive", *Information and Computation*, vol. 193/1, pp. 21-56, 2004.

[Barendregt et al. (1983)] Barendregt H., Coppo M., and Dezani-Ciancaglini M. "A Filter Lambda Model and the Completeness of Type Assignment", *Journal of Symbolic Logic*, 48(4):931-940, 1983.

[Coppo et al. (1983)] Coppo M., Dezani-Ciancaglini M., Honsell F., and Longo G. "Extended Type Structures and Filter Lambda Models", *Logic Colloquium '82*, pp. 241-262, North Holland, 1983.

[van Bakel et al. (1997)] van Bakel S., Liquori L., Ronchi Della Rocca S., and Urzyczyn P. "Comparing Cubes of Typed and Type Assignment systems", *Annals of Pure and Applied Logic*, 86(3):267–303, 1997.

[Capitani and Venneri (2001)]
Capitani B., Venneri B. “Hyperformulæ, Parallel Deductions and Intersection-Types”, Proc. BOTH 2001, ENTCS, 50(2):180-198, 2001.

[Coppo and Dezani-Ciancaglini (1980)] Coppo M., Dezani-Ciancaglini M. “An Extension of the Basic Functionality Theory for the λ-calculus”, *Notre Dame J. Formal Logic*, 21(4):685–693, 1980.

[Dezani-Ciancaglini et al. (1998)] Dezani-Ciancaglini M., Giovannetti E., de’ Liguoro U. “Intersection-types, Lambda-models and Böhm Trees”. In MSJ-Memoir Vol. 2 *Theories of Types and Proofs*, volume 2, pp. 45-97. Mathematical Society of Japan, 1998.

[Giannini et al. (1993)] Giannini P., Honsell F., and Ronchi Della Rocca S. “Type Inference: Some Results, Some Problems”, *Fundamenta Informaticæ*, 19(1,2):87–126, 1993.

[Girard (1986)] Girard, J.Y. “The System F of Variable Types, Fifteen years later”, *Theoretical Computer Science*, 45:159–192, 1986.

[Hindley (1984)] Hindley J. R. “Coppo Dezani Types do not Correspond to Propositional Logic”, *Theoretical Computer Science*, 28(1-2):235-236, 1984.

[Howard (1980)] Howard, W. A. “The Formulae-as-Types Notion of Construction”, in: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pp. 479–490, Academic Press, London, 1980.

[Krivine (1990)] Krivine J.L. “Lambda-calcul, Types et Modèles”, *Masson*, 1990.

[Leivant (1983)] Leivant, D. “Polymorphic Type Inference”, *Proc. of POPL*, pp. 88–98, ACM Press, 1983.

[Liquori (1996)] Liquori, L. “Type Assigment Systems for Lambda Calculi and for the Lambda Calculus of Objects”, Ph.D. thesis, 193 pp., University of Turin, 1996.

[Pottinger (1980)] Pottinger G. “A Type Assignment for the Strongly Normalizable λ-terms”, in: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pp. 561-577, Academic Press, London, 1980.

[Pierce and Turner (1994)] Pierce, B., C. and Turner, D., N. “Simple Type-theoretic Foundations for Object-oriented Programming”, *Journal of Functional Programming*, 4(2):207–247, 1994.

[Reynolds (1996)] Reynolds J.C. “Design of the Programming Language Forsythe”, in: *Algol-like Languages*, O’Hearn and Tennent ed.s, Birkhauser, 1996.

[Ronchi Della Rocca (2002)] Ronchi Della Rocca S. “Intersection-Typed Lambda-Calculus", In *Proc of ICTRS*, ENTCS, 70(1), 2002.

[Ronchi Della Rocca and Roversi (2001)] Ronchi Della Rocca S. and Roversi L. “Intersection Logic”, *Proc. of CSL*, LNCS 2142, pp. 414-428, Springer-Verlag, 2001.

[Wells et al. (2002)] Wells J.B., Dimock A., Muller R., and Turbak F., “A Calculus with Polymorphic and Polyvariant Flow Types”, *Journal of Functional Programming*, 12(3), pp. 183-227, 2002.

[Wells and Haack (2006)] Wells J.B., and Haack C. “Branching Types”, *Information and Computation*, 2006, To appear.