Logic and computation in a lambda calculus with intersection and union types

Daniel J. Dougherty
Worcester Polytechnic Institute, USA

Luigi Liquori
INRIA, France

Abstract

We present an explicitly typed lambda calculus "*à la* Church" based on the union and intersection types discipline; this system is the counterpart of the standard type assignment calculus "*à la* Curry." Our typed calculus enjoys Subject Reduction and confluence, and typed terms are strongly normalizing when the universal type is omitted. Moreover both type checking and type reconstruction are decidable. In contrast to other typed calculi, a system with union types will fail to be "coherent" in the sense of Tannen, Coquand, Gunter, and Scedrov: different proofs of the same typing judgment will not necessarily have the same meaning. In response, we introduce a decidable notion of equality on type-assignment derivations inspired by the equational theory of bicartesian-closed categories.

1 Introduction

We address the problem of designing a λ-calculus *à la* Church corresponding to Curry-style type assignment to an untyped λ-calculus with intersection and union types [16, 3]. In particular, we define a typed language such that its relationship with the intersection-union type assignment system fulfills the following *desiderata*: (i) typed and type assignment derivations are *isomorphic*, *i.e.*, the application of an *erasing function* 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; (ii) type checking and type reconstruction are decidable; (iii) reduction on typed terms has the same fundamental nice properties of reduction on terms receiving a type in the type-assignment system, such as confluence, preservation of typing under reduction, and strong normalization of terms typable without the universal type ω.

The challenges in defining such a calculus are already present in the context of intersection types, as evidenced by the polymorphic identity, with the following type-derivation in Curry style:

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

This is untypable using a naïve corresponding rule *à la* Church for the introduction of intersection types:

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

A solution to this problem was introduced in [15] where a calculus is designed whose terms comprise two parts, carrying computational and logical information respectively. The first component (the *marked-term*) is a simply typed λ-term, but types are variable-marks. 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 seen as an encoding of a fragment

of intuitionistic logic; it codifies a set of proofs that is strictly bigger than those corresponding to intersection type derivations (see [23]). There are other proposals in the literature for a λ-calculus typed with intersection types [17, 20, 4, 26, 22]. The languages proposed in these papers have been designed with various purposes, and they do not satisfy one or more of our desiderata above. A fuller discussion of this related work can be found in [15].

In this paper we extend the system of [15] to a calculus with union types. This is non-trivial, essentially because the standard typing rule for $\vee$-elimination is, as has been noted by many authors, so awkward. The difficulty manifests itself primarily in the (necessary) complexity of the definition of β-reduction on typed terms (see Section 5.2). On the other hand our solution exposes an interesting duality between the techniques required for intersections and for unions (Remark 5.2.1). Our typed reduction is well-behaved: it confluent, obeys subject reduction, and is strongly normalizing on terms typed with the universal type. But it must be taken on its own terms, not as a commentary on the untyped system.

Beyond solving the technical problem of extending the proof-term technique to handle union types, this paper makes a contribution to the study of the semantics of typed calculi viewed as foundations for typed programming language with unions, specifically to the investigation of *coherence*. In a typed programming language typing is an integral part of the semantics of a term. Indeed, the meaning of a typed term is not a function of the raw term but rather of the *typing judgment* of which the term is the subject. Reynolds [21] has dubbed this the *intrinsic* approach to semantics, as opposed to the *extrinsic* semantics given to terms in a type assignment system.

Now, in many type systems typing judgments can be derived in several ways, so the question of the relationship between the meanings of these judgments arises naturally. This question has been addressed in the literature in several settings, including languages with subtyping and generic operators [18], languages with subtyping and intersection types [19], and languages with polymorphism and recursive types [24, 8]. The answer in all these cases has been, "all derivations of the same type judgment have the same meaning." Following [24] this phenomenon has come to be called *coherence*. In the cited work judgments take their meaning in categories where intersections are modeled as categorical *products*: for a discussion of this point see [21] Section 16.6.

But coherence fails for a language with union types, if unions are modeled in the natural way as categorical coproducts. As a simple example, let σ be any type and consider the judgment $\vdash \lambda x.x : (\sigma \to \sigma) \vee (\sigma \to \sigma)$. There are obviously two derivations of this judgment, one corresponding to injection "from the left" and the other to injection "from the right." No reasonable semantics will equate these injections: it is an easy exercise to show that for any σ, if the two injections from σ to $(\sigma \vee \sigma)$ are equal, then any two arrows with source σ will be equal.

So the coherence question requires new analysis in the presence of union types. In this paper we reformulate the question as, "when are two different derivations of the same typing judgment equal?" (Cf. the discussion in [14], page 117, of the coherence problem for monoidal categories.) In Section 6 we show decidability of coherence under two important "type theories" (in the sense of [3]).

The failure of coherence has consequences for reduction of typed terms. In an intrinsic semantics the meaning of a term is a function of its type-derivation. Since reduction must, above all else, respect semantics, it follows that reduction should "respect" the type-derivation. When the language is coherent this is no constraint, and reduction can be defined purely in terms of the raw term that is the subject of the typing judgment. Thus, in typical typed calculi, reduction on typed terms is simply β-reduction, "ignoring the types." But in a system where coherence fails it is crucially important that reduction avoid the blunder of reducing a typed term and failing to preserve the semantics of the term's type-derivation. In the system presented in this paper this condition is reflected in the rather complex definition of reduction in Section 5 and in the fact that typed reduction can even "get stuck" relative to untyped reduction. For similar reasons the Subject Expansion property fails even though the type system has a universal type ω.

Some details have been omitted here for lack of space (see also the web appendix at authors web

pages `http://web.cs.wpi.edu/~dd/publications` and `http://www-sop.inria.fr/members/Luigi.Liquori/PAPERS`); familiarity with [3] and with [15] will be helpful; the latter paper is a good source of examples restricted to the intersection types setting.

2 Intersection and Union Types

2.1 $\Lambda_{\mathsf{u}}^{\wedge\vee}$: Curry-style type assignment with intersections and unions

The set Λ is the set of untyped terms of the λ-calculus:

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

We consider terms modulo α-conversion. Capture-avoiding substitution $M[N/x]$ of term N for variable x into term M is defined in the usual way. The reduction relation $\rightarrow_\beta$ is defined on untyped terms as the compatible closure of the relation $(\lambda x.M)N \rightarrow_\beta M[N/x]$.

Fix a set $\mathscr{V}$ of *type variables* and let ω be a distinguished *type constant.* The set $\mathscr{T}$ of *types* is generated from $\mathscr{V}$ and ω by the binary constructors $\rightarrow, \wedge$, and $\vee$. We use lowercase Greek letters to range over types.

Definition 1. *The Intersection-Union Type Assignment System $\Lambda_{\mathsf{u}}^{\wedge\vee}$ is the set of inference rules in Figure 1 for assigning intersection and union types to terms of the untyped λ-calculus.*

Let $B \triangleq \{x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n\}$ ($i \neq j$ implies $x_i \not\equiv x_j$), and $B, x{:}\sigma \triangleq B \cup \{x{:}\sigma\}$

$$\frac{}{B \vdash M : \omega}\ (\omega) \qquad \frac{x{:}\sigma \in B}{B \vdash x : \sigma}\ (Var)$$

$$\frac{B, x{:}\sigma_1 \vdash M : \sigma_2}{B \vdash \lambda x.M : \sigma_1 \rightarrow \sigma_2}\ (\rightarrow I) \qquad \frac{B \vdash M : \sigma_1 \rightarrow \sigma_2 \quad B \vdash N : \sigma_1}{B \vdash MN : \sigma_2}\ (\rightarrow E)$$

$$\frac{B \vdash M : \sigma_1 \quad B \vdash M : \sigma_2}{B \vdash M : \sigma_1 \wedge \sigma_2}\ (\wedge I) \qquad \frac{B \vdash M : \sigma_1 \wedge \sigma_2 \quad i = 1,2}{B \vdash M : \sigma_i}\ (\wedge E_i)$$

$$\frac{B \vdash M : \sigma_i \quad i = 1,2}{B \vdash M : \sigma_1 \vee \sigma_2}\ (\vee I_i) \qquad \frac{B, x{:}\sigma_1 \vdash M : \sigma_3 \quad B, x{:}\sigma_2 \vdash M : \sigma_3 \quad B \vdash N : \sigma_1 \vee \sigma_2}{B \vdash M[N/x] : \sigma_3}\ (\vee E)$$

Figure 1: The Intersection-Union Type Assignment System $\Lambda_{\mathsf{u}}^{\wedge\vee}$

Here are two crucial properties of the system $\Lambda_{\mathsf{u}}^{\wedge\vee}$.

Theorem 1. *[3]*

- *The terms typable without use of the ω rule are precisely the strongly normalizing terms.*
- *If $B \vdash M : \sigma$ and $M \rightarrow_{gk} N$ then $B \vdash N : \sigma$. Here $\rightarrow_{gk}$ is the well-known "Gross-Knuth" parallel reduction [13].*

2.2 $\Lambda_{\mathsf{t}}^{\wedge\vee}$: Church-style typing with intersections and unions

The key idea in the design of the intersection-union 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 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.

2.3 The Proof-term calculus $\Lambda\mathscr{P}^{\wedge\vee}$

The terms of $\Lambda\mathscr{P}^{\wedge\vee}$ are encodings, via the Curry-Howard isomorphism, of the proofs of type-assignment derivations. 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 a derivation through an association between variable-marks and types.

Definition 2. *Fix a set of variable-marks ι. The raw terms of $\Lambda\mathscr{P}^{\wedge\vee}$ are given as follows:*

$$\Delta \quad ::= \quad \iota \mid * \mid \lambda\iota{:}\sigma.\Delta \mid \Delta\Delta \mid \langle \Delta,\, \Delta\rangle \mid [\Delta,\, \Delta] \mid \mathsf{pr_i}\Delta \mid \mathsf{in_i}\Delta \qquad i = 1,2$$

The $\Lambda\mathscr{P}^{\wedge\vee}$ calculus works modulo α-conversion (denoted by $=_\alpha$) defined as usual. Capture-avoiding substitution of the proof-term Δ_2 for variable ι in term Δ_1 is denoted $\Delta_1[\Delta_2/\iota]$.

Definition 3. *The typing judgments for proof-terms $\Lambda\mathscr{P}^{\wedge\vee}$ are defined by the rules in Figure 2.*

Let $G \triangleq \{\iota_1{:}\sigma_1, \ldots, \iota_n{:}\sigma_n\}$ ($i \neq j$ implies $\iota_i \not\equiv \iota_j$), and $G, \iota{:}\sigma \triangleq G \cup \{\iota{:}\sigma\}$

$$\frac{}{G \vdash * : \omega}\ (\omega) \qquad \frac{\iota{:}\sigma \in G}{G \vdash \iota : \sigma}\ (Var)$$

$$\frac{G, \iota{:}\sigma_1 \vdash \Delta : \sigma_2}{G \vdash \lambda\iota{:}\sigma_1.\Delta : \sigma_1 \to \sigma_2}\ (\to I) \qquad \frac{G \vdash \Delta_1 : \sigma_1 \to \sigma_2 \quad G \vdash \Delta_2 : \sigma_1}{G \vdash \Delta_1\Delta_2 : \sigma_2}\ (\to E)$$

$$\frac{G \vdash \Delta_1 : \sigma_1 \quad G \vdash \Delta_2 : \sigma_2}{G \vdash \langle \Delta_1\,,\, \Delta_2\rangle : \sigma_1 \wedge \sigma_2}\ (\wedge I) \qquad \frac{G \vdash \Delta : \sigma_1 \wedge \sigma_2 \quad i = 1,2}{G \vdash \mathsf{pr_i}\Delta : \sigma_i}\ (\wedge E_i)$$

$$\frac{G \vdash \Delta : \sigma_i \quad i = 1,2}{G \vdash \mathsf{in_i}\Delta : \sigma_1 \vee \sigma_2}\ (\vee I_i) \qquad \frac{G, \iota_1{:}\sigma_1 \vdash \Delta_1 : \sigma_3 \quad G, \iota_2{:}\sigma_2 \vdash \Delta_2 : \sigma_3 \quad G \vdash \Delta_3 : \sigma_1 \vee \sigma_2}{G \vdash [\lambda\iota_1{:}\sigma_1.\Delta_1\,,\, \lambda\iota_2{:}\sigma_2.\Delta_1]\Delta_3 : \sigma_3}\ (\vee E)$$

Figure 2: The type system for the proof calculus $\Lambda\mathscr{P}^{\wedge\vee}$

Since $\Lambda\mathscr{P}^{\wedge\vee}$ is a simply-typed λ-calculus it can naturally be interpreted in cartesian closed categories. A term $[\Delta_1\,,\, \Delta_2]$ corresponds to the "co-pairing" of two arrows Δ_i to build an arrow out of a coproduct type. Then the term $[\lambda\iota_1{:}\sigma_1.\Delta_1\,,\, \lambda\iota_2{:}\sigma_2.\Delta_1]\Delta_3$ corresponds to the familiar `case` statement.

The type ω plays the role of a terminal object, that is to say it is an object with a single element. The connection with type-assignment is this: every term can be assigned type ω so all "proofs" of that judgment have no content: all these proofs are considered identical ([21], page 372). It is typical to name the unique element of the terminal object as $*$. This explains the typing rule for $*$ in Figure 2.

There is a natural equality theory for the terms $\Lambda\mathscr{P}^{\wedge\vee}$; we record it now and will return to it in Section 5.

Definition 4. *The equational theory $\cong$ on proof-terms is defined by the following axioms (we assume that in each equation the two sides have the same type).*

$$(\lambda\iota{:}\sigma.\Delta_1)\,\Delta_2 = \Delta_1[\Delta_2/\iota] \tag{1}$$
$$\mathsf{pr}_i\langle\Delta_1\,,\,\Delta_2\rangle = \Delta_i \qquad i=1,2 \tag{2}$$
$$[\lambda\iota_1{:}\sigma_1.\Delta_1\,,\,\lambda\iota_2{:}\sigma_2.\Delta_2]\,\mathsf{in}_i\Delta = \Delta_i[\Delta/\iota] \qquad i=1,2 \tag{3}$$
$$\lambda\iota{:}\sigma_1.\Delta\,\iota = \Delta \qquad \iota\notin\mathsf{Fv}(\Delta) \tag{4}$$
$$\langle\mathsf{pr}_1\Delta,\,\mathsf{pr}_2\Delta\rangle = \Delta \tag{5}$$
$$[\lambda\iota{:}\sigma_1.\Delta(\mathsf{in}_1\iota)\,,\,\lambda\iota{:}\sigma_2.\Delta(\mathsf{in}_2\iota)](\iota) = \Delta(\iota) \tag{6}$$
$$\Delta = * \qquad \textit{at type } \omega \tag{7}$$

The first three equations are the familiar "computational" axioms for the arrow, product, and sum data-types. The next four equation capture various "uniqueness" criteria which induce the $\wedge$, $\vee$, and ω type constructors to behave as categorical products, coproducts, and terminal object. The terminal type acts as an empty product; in terms of the proof theory this corresponds to saying that ω admits a unique proof, and is reflected in Equation 7, which says that all proofs of type ω are equal to $*$.

2.4 Typed terms with intersections and unions

Definition 5. *Fix a set of variable-marks ι. The set of* marked-terms *are given as follows:*

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

The set of $\Lambda_{\mathsf{t}}^{\wedge\vee}$ of typed terms *is the set of expressions $M@\Delta$ where M is a marked-term and Δ is a proof-term.*

As usual we consider terms modulo renaming of bound variables. Formally this is defined via the notion of α-conversion, which requires some extra care in our setting, so we give the definition explicitly:

Definition 6 (α-conversion)**.** *The α-conversion, denoted by $=_\alpha$, on well formed terms can be defined as the symmetric, transitive, reflexive, and contextual closure of :*

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

Definition 7 (Church-style typing)**.** *The typing rules are presented in Figure 3. The system proves judgments of the shape $\Gamma\vdash M@\Delta:\sigma$, where Γ is a context and $M@\Delta$ is a typed term.*

Intuitively: in the judgment, the type-context Γ assigns union 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 $\Lambda_{\mathsf{t}}^{\wedge\vee}$. The proof-term keeps track of the type of the used mark together with a trace of the *skeleton* of the derivation tree. The proof-term Δ plays the role of a road map to backtrack (*i.e.* roll back) the derivation tree.

2.5 Example of typing for $\Lambda_{\mathsf{t}}^{\wedge\vee}$

The reader will find a good number of examples showing some typing in the intersection type system in [15]. As an example of the present system using intersection and union types in an essential way, we treat the example (due to Pierce) that shows the failure of subject reduction for simple, non parallel, reduction in [3]. Let I denote the identity. Then, the untyped (parallel) reduction is: $x\,(\mathsf{I}\,(yz))\,(\mathsf{I}\,(yz)) \Rightarrow_\beta$

Let $\Gamma \stackrel{\triangle}{=} \{x_1@\iota_1{:}\sigma_1, \ldots, x_n@\iota_2{:}\sigma_n\}$ ($i \neq j$ implies $x_i \not\equiv x_j$), and $\Gamma, x@\iota{:}\sigma \stackrel{\triangle}{=} \Gamma \cup \{x@\iota{:}\sigma\}$

$$\frac{}{\Gamma \vdash M@* : \omega}\ (\omega) \qquad \frac{x@\iota{:}\sigma \in \Gamma}{\Gamma \vdash x@\iota : \sigma}\ (Var)$$

$$\frac{\Gamma, x@\iota{:}\sigma_1 \vdash M@\Delta : \sigma_2}{\Gamma \vdash \lambda x{:}\iota.M@\lambda\iota{:}\sigma_1.\Delta : \sigma_1 \to \sigma_2}\ (\to I) \qquad \frac{\Gamma \vdash M@\Delta_1 : \sigma_1 \to \sigma_2 \quad \Gamma \vdash N@\Delta_2 : \sigma_1}{\Gamma \vdash MN@\Delta_1\,\Delta_2 : \sigma_2}\ (\to E)$$

$$\frac{\Gamma \vdash M@\Delta_1 : \sigma_1 \quad \Gamma \vdash M@\Delta_2 : \sigma_2}{\Gamma \vdash M@\langle \Delta_1\,,\, \Delta_2\rangle : \sigma_1 \wedge \sigma_2}\ (\wedge I) \qquad \frac{\Gamma \vdash M@\Delta : \sigma_1 \wedge \sigma_2 \quad i = 1,2}{\Gamma \vdash M@\mathsf{pr_i}\Delta : \sigma_i}\ (\wedge E_i)$$

$$\frac{\Gamma \vdash M@\Delta : \sigma_i \quad i = 1,2}{\Gamma \vdash M@\mathsf{in_i}\Delta : \sigma_1 \vee \sigma_2}\ (\vee I_i) \qquad \frac{\begin{array}{l}\Gamma, x@\iota_1{:}\sigma_1 \vdash M@\Delta_1 : \sigma_3 \\ \Gamma, x@\iota_2{:}\sigma_2 \vdash M@\Delta_2 : \sigma_3 \quad \Gamma \vdash N@\Delta_3 : \sigma_1 \vee \sigma_2\end{array}}{\Gamma \vdash M[N/x]@[\lambda\iota_1{:}\sigma_1.\Delta_1\,,\, \lambda\iota_2{:}\sigma_2.\Delta_2]\,\Delta_3 : \sigma_3}\ (\vee E)$$

Figure 3: The type system for the typed calculus $\Lambda_{\mathsf{t}}^{\wedge\vee}$

$x\,(yz)\,(yz)$. Under the type context $B \stackrel{\triangle}{=} x{:}(\sigma_1 \to \sigma_1 \to \tau) \wedge (\sigma_2 \to \sigma_2 \to \tau), y{:}\rho \to \sigma_1 \vee \sigma_2, z : \rho$, the redex can be typed as follows (the derivation for the reductum being simpler):

$$\dfrac{\dfrac{\dfrac{B, w{:}\sigma_1 \vdash x : \sigma_1 \to \sigma_1 \to \tau \quad B, w{:}\sigma_1 \vdash w : \sigma_1}{B, w{:}\sigma_1 \vdash xw : \sigma_1 \to \tau \quad B, w{:}\sigma_1 \vdash w : \sigma_1}}{B, w{:}\sigma_1 \vdash xww : \tau} \quad \dfrac{\dfrac{B, w{:}\sigma_2 \vdash x : \sigma_2 \to \sigma_2 \to \tau \quad B, w{:}\sigma_2 \vdash w : \sigma_2}{B, w{:}\sigma_2 \vdash xw : \sigma_2 \to \tau \quad B, w{:}\sigma_2 \vdash w : \sigma_2}}{B, w{:}\sigma_2 \vdash xww : \tau} \quad \dfrac{B \vdash \mathsf{I} : \sigma_1 \vee \sigma_2 \to \sigma_1 \vee \sigma_2 \quad B \vdash yz : \sigma_1 \vee \sigma_2}{B \vdash \mathsf{I}\,(yz) : \sigma_1 \vee \sigma_2}}{B \vdash x\,(\mathsf{I}\,(yz))\,(\mathsf{I}\,(yz)) : \tau}\ (\vee E)$$

We look now for the corresponding typed derivations. The corresponding typed term of $x\,(\mathsf{I}\,(yz))\,(\mathsf{I}\,(yz))$ is

$$x(\underbrace{(\lambda v{:}\iota_3.v)}_{\mathsf{I_t}}(yz))\,(\underbrace{(\lambda v{:}\iota_3.v)}_{\mathsf{I_t}}(yz))@[\underbrace{\lambda\iota_1{:}\sigma_1.(\mathsf{pr}_1\iota)\,\iota_1\,\iota_1}_{\Delta_1}\,,\,\underbrace{\lambda\iota_2{:}\sigma_2.(\mathsf{pr}_2\iota)\,\iota_2\,\iota_2}_{\Delta_2}]\,(\underbrace{(\lambda\iota_3{:}\sigma_1 \vee \sigma_2.\iota_3)}_{\Delta_3}(\iota_4\,\iota_5))$$

Under the type context $\Gamma \stackrel{\triangle}{=} x@\iota{:}(\sigma_1 \to \sigma_1 \to \tau) \wedge (\sigma_2 \to \sigma_2 \to \tau), y@\iota_4{:}\rho \to \sigma_1 \vee \sigma_2, z@\iota_5 : \rho$, and $\Gamma_1 = \Gamma, w@\iota_1{:}\sigma_1$ and $\Gamma_2 = \Gamma, w@\iota_2{:}\sigma_2$, the above term can be typed as follows:

$$\dfrac{\dfrac{\dfrac{\Gamma_1 \vdash x@\mathsf{pr}_1\iota : \sigma_1 \to \sigma_1 \to \tau \quad \Gamma_1 \vdash w@\iota_1 : \sigma_1}{\Gamma_1 \vdash xw@(\mathsf{pr}_1\iota)\,\iota_1 : \sigma_1 \to \tau \quad \Gamma_1 \vdash w@\iota_1 : \sigma_1}}{\Gamma_1 \vdash xww@(\mathsf{pr}_1\iota)\,\iota_1\,\iota_1 : \tau} \quad \dfrac{\dfrac{\Gamma_2 \vdash x@\mathsf{pr}_2\iota : \sigma_2 \to \sigma_2 \to \tau \quad \Gamma_2 \vdash w@\iota_2 : \sigma_2}{\Gamma_2 \vdash xw@(\mathsf{pr}_2\iota)\,\iota_2 : \sigma_2 \to \tau \quad \Gamma_2 \vdash w@\iota_2 : \sigma_2}}{\Gamma_2 \vdash xww@(\mathsf{pr}_2\iota)\,\iota_2\,\iota_2 : \tau} \quad \dfrac{\Gamma \vdash \mathsf{I_t}@\Delta_3 : \sigma_1 \vee \sigma_2 \to \sigma_1 \vee \sigma_2 \quad \Gamma \vdash yz@\iota_4\,\iota_5 : \sigma_1 \vee \sigma_2}{\Gamma \vdash \mathsf{I_t}\,(yz)@(\Delta_3\,(\iota_4\,\iota_5)) : \sigma_1 \vee \sigma_2}}{\Gamma \vdash x\,(\mathsf{I_t}\,(yz))\,(\mathsf{I_t}\,(yz))@[\Delta_1\,,\,\Delta_2]\,(\Delta_3\,(\iota_4\,\iota_5)) : \tau}$$

3 The Isomorphism between $\Lambda_{\mathsf{u}}^{\wedge\vee}$ and $\Lambda_{\mathsf{t}}^{\wedge\vee}$

In this section we prove that the type system for $\Lambda_{\mathsf{t}}^{\wedge\vee}$ is isomorphic to the classical system for $\Lambda_{\mathsf{u}}^{\wedge\vee}$ of [3]. The isomorphism is given for a customization of the general definition of isomorphism given in [25], to the case of union types and proof-terms.

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).

$$\mathscr{F}\left(\dfrac{\mathscr{D}_1^{\mathsf{t}} : \Gamma \vdash M@\Delta_1 : \sigma_1 \quad \mathscr{D}_2^{\mathsf{t}} : \Gamma \vdash M@\Delta_2 : \sigma_2}{\Gamma \vdash M@\langle \Delta_1 , \Delta_2\rangle : \sigma_1 \wedge \sigma_2}\ (\wedge I)\right) \triangleq \begin{cases} \dfrac{\mathscr{F}(\mathscr{D}_1^{\mathsf{t}}) : B \vdash M' : \sigma_1 \quad \mathscr{F}(\mathscr{D}_2^{\mathsf{t}}) : B \vdash M' : \sigma_2}{B \vdash M' : \sigma_1 \wedge \sigma_2}\ (\wedge I) \\ \mathscr{E}(\Gamma) = B \ \&\ \mathscr{E}(M@\langle \Delta_1 , \Delta_2\rangle) = M' \end{cases}$$

$$\mathscr{F}\left(\dfrac{\mathscr{D}_1^{\mathsf{t}} : \Gamma, x@\iota_1{:}\sigma_1 \vdash M@\Delta_1 : \sigma_3 \quad \mathscr{D}_2^{\mathsf{t}} : \Gamma, x@\iota_2{:}\sigma_2 \vdash M@\Delta_2 : \sigma_3 \quad \mathscr{D}_3^{\mathsf{t}} : \Gamma \vdash N@\Delta_3 : \sigma_1 \vee \sigma_2}{\Gamma \vdash M[N/x]@\left[\begin{array}{l}\lambda\iota_1{:}\sigma_1.\Delta_1,\\ \lambda\iota_2{:}\sigma_2.\Delta_2\end{array}\right]\Delta_3 : \sigma_3}\ (\vee E)\right) \triangleq \begin{cases} \dfrac{\mathscr{F}(\mathscr{D}_1^{\mathsf{t}}) : B, x{:}\sigma_1 \vdash M'' : \sigma_3 \quad \mathscr{F}(\mathscr{D}_2^{\mathsf{t}}) : B, x{:}\sigma_2 \vdash M'' : \sigma_3 \quad \mathscr{F}(\mathscr{D}_3^{\mathsf{t}}) : B \vdash N' : \sigma_1 \vee \sigma_2}{B \vdash M' : \sigma_3}\ (\vee E) \\ \mathscr{E}(\Gamma) = B \ \&\ \mathscr{E}(M[N/x]@\left[\begin{array}{l}\lambda\iota_1{:}\sigma_1.\Delta_1,\\ \lambda\iota_2{:}\sigma_2.\Delta_2\end{array}\right]\Delta_3) = M' \\ \mathscr{E}(M@\Delta_{1,2}) = M'' \ \&\ \mathscr{E}(N@\Delta_3) = N' \end{cases}$$

$$\mathscr{G}\left(\dfrac{\mathscr{D}_1^{\mathsf{u}} : B \vdash M' : \sigma_1 \quad \mathscr{D}_2^{\mathsf{u}} : B \vdash M' : \sigma_2}{B \vdash M' : \sigma_1 \wedge \sigma_2}\ (\wedge I)\right) \triangleq \begin{cases} \dfrac{\mathscr{G}(\mathscr{D}_1^{\mathsf{u}}) : \Gamma \vdash M@\Delta_1 : \sigma_1 \quad \mathscr{G}(\mathscr{D}_2^{\mathsf{u}}) : \Gamma \vdash M@\Delta_2 : \sigma_2}{\Gamma \vdash M@\langle \Delta_1 , \Delta_2\rangle : \sigma_1 \wedge \sigma_2}\ (\wedge I) \\ \mathscr{E}(\Gamma) = B \ \&\ \mathscr{E}(M@\langle \Delta_1 , \Delta_2\rangle) = M' \end{cases}$$

$$\mathscr{G}\left(\dfrac{\mathscr{D}_1^{\mathsf{u}} : B, x{:}\sigma_1 \vdash M' : \sigma_3 \quad \mathscr{D}_2^{\mathsf{u}} : B, x{:}\sigma_2 \vdash M' : \sigma_3 \quad \mathscr{D}_3^{\mathsf{u}} : B \vdash N' : \sigma_1 \vee \sigma_2}{B \vdash M'[N'/x] : \sigma_3}\ (\vee E)\right) \triangleq \begin{cases} \dfrac{\mathscr{G}(\mathscr{D}_1^{\mathsf{u}}) : \Gamma, x@\iota_1{:}\sigma_1 \vdash M@\Delta_1 : \sigma_3 \quad \mathscr{G}(\mathscr{D}_2^{\mathsf{u}}) : \Gamma, x@\iota_2{:}\sigma_2 \vdash M@\Delta_2 : \sigma_3 \quad \mathscr{G}(\mathscr{D}_3^{\mathsf{u}}) : \Gamma \vdash N@\Delta_3 : \sigma_1 \vee \sigma_2}{\Gamma \vdash M[N/x]@\left[\begin{array}{l}\lambda\iota_1{:}\sigma_1.\Delta_1,\\ \lambda\iota_2{:}\sigma_2.\Delta_2\end{array}\right]\Delta_3 : \sigma_3}\ (\vee E) \\ \mathscr{E}(\Gamma) = B \ \&\ \mathscr{E}(M[N/x]@\left[\begin{array}{l}\lambda\iota_1{:}\sigma_1.\Delta_1,\\ \lambda\iota_2{:}\sigma_2.\Delta_2\end{array}\right]\Delta_3) = M'[N'/x] \\ \mathscr{E}(M@\Delta_{1,2}) = M' \ \&\ \mathscr{E}(N@\Delta_3) = N' \end{cases}$$

Figure 4: The Functions $\mathscr{F}$ and $\mathscr{G}$ (sketch).

Definition 8 (Church *vs.* Curry)**.**

1. *The type-erasing function $\mathscr{E} : \Lambda_{\mathsf{t}}^{\wedge\vee} \Rightarrow \Lambda$ is inductively defined on terms as follows:*

$$\mathscr{E}(x@\_) \triangleq x \qquad \mathscr{E}(\lambda x{:}\iota.M@\_) \triangleq \lambda x.\mathscr{E}(M@\_) \qquad \mathscr{E}(MN@\_) \triangleq \mathscr{E}(M@\_)\,\mathscr{E}(N@\_)$$

$\mathscr{E}$ is pointwise extended to contexts in the obvious way.

$$
\begin{array}{lcll}
\mathsf{Type}(\Gamma, M@\Delta) & \stackrel{\triangle}{=} & \text{match } M@\Delta \text{ with} & \\
\_@* & \Rightarrow & \omega & \\
\_@\mathsf{pr}_i\Delta_1 & \Rightarrow & \sigma_i \quad i=1,2 & \text{if } \mathsf{Type}(\Gamma, M@\Delta_1) = \sigma_1 \wedge \sigma_2 \\
\_@\langle \Delta_1\,,\, \Delta_2\rangle & \Rightarrow & \sigma_1 \wedge \sigma_2 & \text{if} \quad \mathsf{Type}(\Gamma, M@\Delta_1) = \sigma_1 \\
 & & & \text{and } \mathsf{Type}(\Gamma, M@\Delta_2) = \sigma_2 \\
\_@\mathsf{in}_i\Delta_1 & \Rightarrow & \sigma_1 \vee \sigma_2 & \text{if } \mathsf{Type}(\Gamma, M@\Delta_1) = \sigma_i \quad i=1,2 \\
\_@\left[\begin{array}{l}\lambda\iota_1{:}\sigma_1.\Delta_1,\\ \lambda\iota_2{:}\sigma_2.\Delta_2\end{array}\right]\Delta_3 & \Rightarrow & \sigma_3 & \text{if} \quad \mathsf{Type}((\Gamma, x@\iota_1{:}\sigma_1), M'@\Delta_1) = \sigma_3 \\
 & & & \text{and } \mathsf{Type}((\Gamma, x@\iota_2{:}\sigma_2), M'@\Delta_2) = \sigma_3 \\
 & & & \text{and } \mathsf{Type}(\Gamma, N@\Delta_3) = \sigma_1 \vee \sigma_3 \text{ and} \\
 & & & \text{and } M \equiv M'[N/x] \\
x@\iota & \Rightarrow & \sigma & \text{if } x@\iota{:}\sigma \in \Gamma \\
\lambda x{:}\iota.M_1@\lambda\iota{:}\sigma_1.\Delta_1 & \Rightarrow & \sigma_1 \to \sigma_2 & \text{if } \mathsf{Type}((\Gamma, x@\iota{:}\sigma_1), M_1@\Delta_1) = \sigma_2 \\
M_1\,M_2@\Delta_1\,\Delta_2 & \Rightarrow & \sigma_2 & \text{if} \quad \mathsf{Type}(\Gamma, M_1@\Delta_1) = \sigma_1 \to \sigma_2 \\
 & & & \text{and } \mathsf{Type}(\Gamma, M_2@\Delta_2) = \sigma_1 \\
\_@\_ & \Rightarrow & \mathsf{false} & \text{otherwise}
\end{array}
$$

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

Figure 5: The Type Reconstruction and Type Checking Algorithms for $\Lambda_{\mathsf{t}}^{\wedge\vee}$.

2. *Let $\mathscr{D}er\Lambda_{\mathsf{u}}^{\wedge}$ and $\mathscr{D}er\Lambda_{\mathsf{t}}^{\wedge}$ be the sets of all (un)typed derivations, and let $\mathscr{D}$, and $\mathscr{D}^{\mathsf{u}}$ denote (un)typed derivations, respectively. The functions $\mathscr{F} : \mathscr{D}er\Lambda_{\mathsf{t}}^{\wedge} \Rightarrow \mathscr{D}er\Lambda_{\mathsf{u}}^{\wedge}$ and $\mathscr{G} : \mathscr{D}er\Lambda_{\mathsf{u}}^{\wedge} \Rightarrow \mathscr{D}er\Lambda_{\mathsf{t}}^{\wedge}$ are indicated in Figure 4: for lack of space only some representative cases are shown (the full definition is in the web appendix).*

Theorem 2 (Isomorphism)**.** *The systems $\Lambda_{\mathsf{t}}^{\wedge\vee}$ and $\Lambda_{\mathsf{u}}^{\wedge\vee}$ are isomorphic in the following sense. $\mathscr{F}\circ\mathscr{G}$ is the identity in $\mathscr{D}er\Lambda_{\mathsf{u}}^{\wedge}$ and $\mathscr{G}\circ\mathscr{F}$ is the identity in $\mathscr{D}er\Lambda_{\mathsf{t}}^{\wedge}$ modulo uniform naming of variable-marks. I.e.,*

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

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

Proof. By induction on the structure of derivations. □

4 Type reconstruction and type checking algorithms

The type reconstruction and the type checking algorithms are presented in Figure 5, and the following theorems holds.

Theorem 3 (Type Reconstruction for $\Lambda_{\mathsf{t}}^{\wedge\vee}$)**.**

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

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

Proof. Soundness is proved by induction over the computation of $\mathsf{Type}(\Gamma, M@\Delta)$, $M@\Delta$, while completeness is proved by induction over the computation of $\mathsf{Type}(\Gamma, M@\Delta)\sigma$. □

Theorem 4 (Type Checking for $\Lambda_{\mathsf{t}}^{\wedge\vee}$)**.** $\Gamma \vdash M@\Delta : \sigma$*, if and only if* $\mathsf{Typecheck}(\Gamma, M@\Delta, \sigma) = \mathsf{true}$.

Proof. The $\Rightarrow$ part can be proved using completeness of the type reconstruction algorithm (Theorem 3), while the $\Leftarrow$ part can be proved using soundness of the type reconstruction algorithm. □

Corollary 1 ($\Lambda_{\mathsf{t}}^{\wedge\vee}$ Judgment Decidability)**.** *It is decidable whether* $\Gamma \vdash M@\Delta : \sigma$ *is derivable.*

5 Reduction in $\Lambda_{\mathsf{t}}^{\wedge\vee}$

As we have seen there is natural erasing function from typed $\Lambda_{\mathsf{t}}^{\wedge\vee}$ terms to untyped terms of $\Lambda_{\mathsf{u}}^{\wedge\vee}$. And reduction in the untyped λ-calculus is simply β-reduction. But as we have discussed in the introduction it would be a mistake to conflate typed and untyped reduction, in part due to the failure of coherence. Reduction on typed terms must respect the semantics of the type derivations, which is to say, *reduction on marked-terms must respect the semantics of the proof-terms.* The definition of the relation $\Rightarrow_\beta$ below ensures this. On the other hand it is useful to perform steps that keep the marked-term unchanged but reduce the proof-term, as long as the semantics of the type-derivation encoded by the proof-term is preserved. This is the role of the relation $\Rightarrow_\Delta$.

5.1 Synchronization

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*), $(\to I)$, and $(\to E)$, but they *get disconnected* when we apply the $(\wedge I)$, $(\vee I)$ or $(\wedge E)$ rules, which change the Δ but not the M. This disconnection is "logged" in the Δ via occurrences of operators $\langle -,\ -\rangle$, $[-,\ -]$, pr_i, and in_i. In order to correctly identify the reductions that need to be performed in parallel in order to preserve the correct syntax of the term, we will define the notion of overlapping. Namely a redex is defining taking into account the surrounding context.

To define β-reduction on typed terms some care is required to manage the variable-marks. For this purpose we view $M@\Delta$ as a pair of trees, so subterms are associated by *tree-addresses*, as usual, sequences of integers.

Definition 9. *For a well-typed* $M@\Delta$*, we define the binary "synchronization" relation* Sync *between tree-addresses in* M *and tree-addresses in* Δ*. The definition is by induction over the typing derivation (see Figure 3). We present a representative set of cases here.*

- *When* $M@\Delta$ *is* $x@\imath$*: we of course take the roots to be related:* $\mathsf{Sync}(\langle\rangle, \langle\rangle)$
- *Case*
$$\frac{\Gamma \vdash M@\Delta_1 : \sigma_1 \to \sigma_2 \quad \Gamma \vdash N@\Delta_2 : \sigma_1}{\Gamma \vdash MN@\Delta_1\,\Delta_2 : \sigma_2}\ (\to E)$$
For an address a *from* MN *and an address* a' *from* $\Delta_1\,\Delta_2$*:* $\mathsf{Sync}(a,a')$ *if and only if either*
  - a *is from* M *and* a' *is from* Δ_1*, that is* $a = 1b$ *and* $a' = 1b'$ *and they were synchronized in* $M@\Delta_1$*, that is* $\mathsf{Sync}(b,b')$*, or*
  - a *and* a' *are from* N *and* Δ_2 *respectively and were synchronized in* $N@\Delta_2$*.*
- *Case*
$$\frac{\Gamma \vdash M@\Delta_1 : \sigma_1 \quad \Gamma \vdash M@\Delta_2 : \sigma_2}{\Gamma \vdash M@\langle\Delta_1\,,\ \Delta_2\rangle : \sigma_1 \wedge \sigma_2}\ (\wedge I)$$
Let a *be an address in* M *and* a' *an address in* $\langle\Delta_1\,,\ \Delta_2\rangle$*. Then* $\mathsf{Sync}(a,a')$ *if and only if* $a' = 1b$ *and* $\mathsf{Sync}(a,b)$ *from* $M@\Delta_1$ *or* $a' = 2b$ *and* $\mathsf{Sync}(a,b)$ *from* $M@\Delta_2$*.*

- *Case*

$$\frac{\Gamma, x@\iota_1{:}\sigma_1 \vdash M@\Delta_1 : \sigma_3 \quad \Gamma, x@\iota_2{:}\sigma_2 \vdash M@\Delta_2 : \sigma_3 \quad \Gamma \vdash N@\Delta_3 : \sigma_1 \vee \sigma_2}{\Gamma \vdash M[N/x]@[\lambda\iota_1{:}\sigma_1.\Delta_1\,,\ \lambda\iota_2{:}\sigma_2.\Delta_2]\,\Delta_3 : \sigma_3} \quad (\vee E)$$

Let a be an address in $M[N/x]$. Then for an address a' from $[\lambda\iota_1{:}\sigma_1.\Delta_1\,,\ \lambda\iota_2{:}\sigma_2.\Delta_2]\,\Delta_3$ we have $\mathsf{Sync}(a,a')$ just in case one of the following holds

  - *a is an address in M other than that of x, and for some i and some address b' of Δ_i we have $\mathsf{Sync}(a,b')$ and a' is the corresponding Δ_i subterm in $[\lambda\iota_1{:}\sigma_1.\Delta_1\,,\ \lambda\iota_2{:}\sigma_2.\Delta_2]\,\Delta_3$ (precisely: $a' = 1ib'$)*
  - *a is an address corresponding to an address b in N after the substitution (precisely, $a = db$ where x occurs at address d in M), a' is an address corresponding to an address b' in Δ_3 ($a' = 2b'$) and we have $\mathsf{Sync}(b,b')$.*

Definition 10. *Consider $M@\Delta$. Let a and b be addresses in M. Say that $a \sim b$ if there is some c an address in Δ with both $\mathsf{Sync}(a,c)$ and $\mathsf{Sync}(b,c)$. In a precisely analogous way we define $\sim$ on addresses in Δ.*

It easy to check that if two addresses are $\sim$ then the corresponding subterms are identical. It is also clear that if $\mathsf{Sync}(a,a')$ and a is the address of a β-redex in M then b is the address of a β-redex in Δ and conversely. It is clear that $\sim$ is an equivalence relation. So we may define: a *synchronized pair* to be a pair (S,S') of sets of addresses in M and Δ respectively such that S and S' are each $\sim$-equivalence classes pointwise related by Sync; that is for each $a \in S$ and each $a' \in S'$ we have $\mathsf{Sync}(a,a')$.

5.2 The reduction relation $\Rightarrow$

We define $\Rightarrow$ as the union of two reductions: $\Rightarrow_\beta$ deals with β-reduction occurring in both the marked- and the proof-term; while $\Rightarrow_\Delta$ deals with reductions arising from proof-term simplifications. Proof-term reduction is defined from the equations in Definition 4.

Definition 11. *The reduction relation $\rightarrow_{\wedge\vee}$ on proof-terms is defined by orienting equations (2), (3), and (7) from Definition 4 from left to right.*

Definition 12.

($\Rightarrow_\beta$) *Let $\mathsf{C}\{\,\}_{i\in I}$ (resp, $\mathsf{C}'\{\,\}_{i\in I}$) be a* multihole *marked-term context (resp. proof-term context), and consider*

$$\mathsf{C}\{\}@\mathsf{C}'\{\}$$

where the indicated occurrences of holes form a synchronized pair of sets of subterms. Then the $\Rightarrow_\beta$ reduction is defined as follows:

$$\mathsf{C}\{(\lambda x{:}\iota.M)\,N\}_{i\in I}@\mathsf{C}'\{(\lambda\iota{:}\sigma_j.\Delta_j)\,\Delta'_j\}_{j\in J} \Rightarrow_\beta \mathsf{C}\{M[N/x]\}_{i\in I}@\mathsf{C}'\{\Delta_j[\Delta'_j/\iota]\}_{j\in J}$$
$$\mathsf{C}\{(\lambda x{:}\iota.M)\,N\}_{i\in I}@\mathsf{C}'\{*\}_{j\in J} \Rightarrow_\beta \mathsf{C}\{M[N/x]\}_{i\in I}@\mathsf{C}'\{*\}_{j\in J}$$

($\Rightarrow_\Delta$) *Let $\mathsf{C}'\{\,\}$ be a plain (monohole) proof-context. Then the $\Rightarrow_\Delta$ reduction is of the form:*

$$M@\mathsf{C}'\{\Delta\} \Rightarrow_\Delta M@\mathsf{C}'\{\Delta'\} \qquad \textit{where } \Delta \rightarrow_{\wedge\vee} \Delta'.$$

Note that the reduction $\Rightarrow_\beta$ is characterized by the two distinct patterns written above. There is no overlap between these two cases, since, as observed just after the definition of $\sim$ a term $(\lambda x{:}\iota.M)\,N$ cannot be synchronized with both an occurrence of $(\lambda\iota{:}\sigma.\Delta)\,\Delta'$ and an occurrence of $*$.

5.2.1 Remarks

- In the definition of $\Rightarrow_\beta$: it is interesting to note the following duality: the typing rule $(\wedge I)$ is what leads us to synchronize one variable-mark ι occurring in a redex in the marked-term (the computation), *e.g.* $(\lambda x{:}\iota.M)\,N$, with potentially many redexes in the proof-term, *e.g.* $(\lambda\iota{:}\sigma_j.\Delta_j)\,\Delta'_j$ with $j \in J$. Symmetrically, the typing rule $(\vee E)$ is what leads us to synchronize one variable-mark occurring in a redex in the logic part, *e.g.* $(\lambda\iota{:}\sigma.\Delta)\,\Delta'$, with potentially many (but equal) redexes in the computational part, *e.g.* i-occurrences of $(\lambda x{:}\iota.M)\,N$ with $i \in I$.
- Implementation of $\Rightarrow$ is potentially complicated by the need to manage the $\sim$ relation. But in an implementation in which subterm-occurrences can be *shared* there is no real need for a "many-to-many" relation on addresses.
- The erasure of the relation $\Rightarrow_\beta$ is similar to (though not identical with) the parallel reduction relation defined in [3].

5.3 Example of reduction for $\Lambda_{\mathsf{t}}^{\wedge\vee}$ (continued)

As an example of the treatment of intersection and union types in our system we examine Pierce's example in [3] showing the failure of subject reduction for simple, non parallel, reduction. The term in question is a good example to show the role of synchronization in reduction on $\Lambda_{\mathsf{t}}^{\wedge\vee}$ terms. Then the complete untyped reduction is:

$$x\,(\mathsf{I}\,(yz))\,(\mathsf{I}\,(yz)) \Rightarrow_\beta \begin{array}{l} \nearrow^\beta x\,(yz)\,(\mathsf{I}\,(yz))\searrow_\beta \\ \searrow_\beta x\,(\mathsf{I}\,(yz))\,(yz)\nearrow^\beta \end{array} x\,(yz)\,(yz).$$

Under the type context $B \triangleq x{:}(\sigma_1 \to \sigma_1 \to \tau) \wedge (\sigma_2 \to \sigma_2 \to \tau), y{:}\rho \to \sigma_1 \vee \sigma_2, z:\rho$, the first two and the last terms can be typed with τ, while terms in the "fork" are not because of the mismatch of the premises in the $(\vee E)$ type assignment rule. Then, the typed term is

$$x(\underbrace{(\lambda v{:}\iota_3.v)}_{\mathsf{I_t}}(yz))\,(\underbrace{(\lambda v{:}\iota_3.v)}_{\mathsf{I_t}}(yz))@[\underbrace{\lambda\iota_1{:}\sigma_1.(\mathsf{pr}_1\iota)\,\iota_1\,\iota_1}_{\Delta_1},\ \underbrace{\lambda\iota_2{:}\sigma_2.(\mathsf{pr}_2\iota)\,\iota_2\,\iota_2}_{\Delta_2}]\,(\underbrace{(\lambda\iota_3{:}\sigma_1.\iota_3)}_{\Delta_3}(\iota_4\,\iota_5))$$

and the typed synchronized reduction goes as follows

$$\begin{array}{ll} x\,(\mathsf{I_t}\,(yz))\,(\mathsf{I_t}\,(yz))@[\Delta_1\,,\,\Delta_2]\,(\Delta_3\,(\iota_4\,\iota_5)) & \Rightarrow_\Delta \\ x\,\underbrace{\overbrace{(\mathsf{I_t}\,(yz))}^{\to_\beta}\,\overbrace{(\mathsf{I_t}\,(yz))}^{\to_\beta}@[\Delta_1\,,\,\Delta_2]\,\overbrace{(\Delta_3\,(\iota_4\,\iota_5))}^{\to_\beta}\,\overbrace{(\Delta_3\,(\iota_4\,\iota_5))}^{\to_\beta}}_{\text{fire a } \Rightarrow_\beta \text{ redex}} & \Rightarrow_\beta x\,(yz)\,(yz)@[\Delta_1\,,\,\Delta_2]\,(\iota_4\,\iota_5)\,(\iota_4\,\iota_5) \end{array}$$

5.4 Properties of $\Rightarrow$

We have seen that the relationship between the corresponding type systems of $\Lambda_{\mathsf{u}}^{\wedge\vee}$ and $\Lambda_{\mathsf{t}}^{\wedge\vee}$ is essentially one of isomorphism. The relationship between the reduction relations in the two calculi is more interesting. First, modulo erasing there is a sense in which $\Rightarrow$ is a sub-relation of untyped $=_\beta$. More precisely:

Lemma 1. *If* $M@\Delta \Rightarrow N@\Delta'$ *then* $\mathscr{E}(M@\Delta) \to \mathscr{E}(N@\Delta')$.

Proof. Straightforward, using the auxiliary result that $\mathscr{E}(M[N/x]@\Delta) \equiv \mathscr{E}(M)[\mathscr{E}(N)/x]$. □

Reduction out of typed terms is well-behaved in the sense witnessed by the following traditional properties.

Theorem 5. *Let $M@\Delta$ be a typable term of $\Lambda_{\mathsf{t}}^{\wedge\vee}$.*

1. *(Subject Reduction) If $\Gamma \vdash M@\Delta : \sigma$ and $M@\Delta \Rightarrow M'@\Delta'$, then $\Gamma \vdash M'@\Delta' : \sigma$.*
2. *(Church-Rosser) The reduction relation $\Rightarrow_\beta$ is confluent out of $M@\Delta$.*
3. *(Strong Normalization) If $M@\Delta$ is a typable without using rule ω then $\Rightarrow_\beta$ is strongly normalizing out of $M@\Delta$.*

Proof. The proof of Subject Reduction is routine. It should be noted that the typical obstacle to Subject Reduction in the presence of a rule such as $(\vee E)$ does not arise for us, since our reduction relation is already necessarily of a "parallel" character due to the requirement of maintaining synchronization. Confluence can be shown by an easy application of standard techniques (for example, the Tait&Martin-Löf parallel reduction argument). Strong Normalization is immediate from the fact that $\rightarrow_{\wedge\vee}$ is strongly normalizing. □

On the other hand we may point out two (related) aspects of typed reduction that are at first glance anomalous.

Getting stuck. The need for marked-term β-redexes to be synchronized with proof-term β-redexes mean that a marked-term β-redex might not be able to participate in a reduction. This can happen when a term $P@[\lambda\iota_1{:}\sigma_1.\Delta_1\,,\ \lambda\iota_2{:}\sigma_2.\Delta_2]\,\Delta_3$ is typed by $(\vee E)$ and the marked-term $P \equiv M[N/x]$ is β-redex. Since the corresponding proof-term $[\lambda\iota_1{:}\sigma_1.\Delta_1\,,\ \lambda\iota_2{:}\sigma_2.\Delta_2]\,\Delta_3$ is not a β-redex in the proof-term calculus we can view the typed term as being "stuck." Now the proof-term may reduce via $\Rightarrow_\Delta$ and eventually become a β-redex. Indeed it is not hard to show that if the term is closed (or if every free variable has Harrop type, defined in Section 6.1) then this will always happen. But in general we can have normal forms in the typed calculus whose erasures contain β-redexes in the sense of untyped λ-calculus. This phenomenon is inherent in having a typed calculus with unions. The β-reductions available in the Curry-style system have a character from the coproduct reductions on proof-terms: a term $[\lambda\iota_1{:}\sigma_1.\Delta_1\,,\ \lambda\iota_2{:}\sigma_2.\Delta_2]\,\Delta_3$ has to wait for its argument Δ_3 to manifest itself as being of the form $\mathsf{in}_i\Delta_4$. And in order to maintain the synchronization between marked-terms and proof-terms, the marked-term β-redex must wait as well.

Another manifestation of the constraint that the marked- and proof- components of a term must be compatible is the fact that—even though the type system has a universal type ω—the system does not have the Subject Expansion property.

Failure of Subject Expansion. There exist typed terms $M@\Delta$ and $M'@\Delta'$ such that $M@\Delta \Rightarrow M'@\Delta'$ and $M'@\Delta'$ is typable but $M@\Delta$ is not typable. For example

$$(\lambda x{:}\iota_1.x)@\mathsf{pr}_1\langle\lambda\iota_1{:}\sigma.\iota_1\,,\ \lambda\iota_1{:}\sigma.\lambda\iota_2{:}\tau.\iota_1\rangle \quad\Rightarrow\quad (\lambda x{:}\iota_1.x)@(\lambda\iota_1{:}\sigma.\iota_1)$$

The latter is clearly a typed term with type $\sigma \to \sigma$. But it is easy to see that in order for the former term to be typed it would have to be the case that $(\lambda x{:}\sigma_1.x)@\mathsf{pr}_1\langle\lambda\iota_1{:}\sigma.\iota_1\,,\ \lambda\iota_1{:}\sigma.\lambda\iota_2{:}\tau.\iota_1\rangle$ is a typed term, which means in turn that $(\lambda x{:}\iota_1.x)@(\lambda\iota_1{:}\sigma.\lambda\iota_2{:}\tau.\iota_1)$ is a typed term; and this is not the case.

Of course in untyped λ-calculus we may use ω to type terms which are "erased" in a reduction: this is the essence of why Subject Expansion holds in the presence of ω. But this move is not available to us here. The problem with $(\lambda x{:}\sigma_1.x)@(\lambda\iota_1{:}\sigma.\lambda\iota_2{:}\tau.\iota_1)$ as a typed term is not the lack of a general-enough type, it is the fact that $(\lambda\iota_1{:}\sigma.\lambda\iota_2{:}\tau.\iota_1)$ cannot encode the shape of a derivation of a type-assignment to $(\lambda x.x)$.

6 Deciding type-derivation equality

As described in the introduction, when semantics is given to typing derivations in Church-style, the question arises: "what is the relationship between the semantics of different derivations of the same typing judgment?" In this section we explore the closely related question "when should two derivations of the same judgment be considered equal?"

We locate the semantics of type-derivations in a cartesian closed category with binary coproducts (*i.e.*, a bicartesian closed category but without an initial type). Since we are interested here in those equalities between derivations which hold in all such categories interpreting the derivations, we focus on the equalities holding in the *free* bi-cartesian closed category over the graph whose objects are the type-variables and the constant ω and whose arrows include the primitive coercion-constants Σ. These equalities are determined by the equational theory $\cong$.

The theory of these equations is surprisingly subtle. On the positive side, it is proved in [10] that a Friedman completeness theorem holds for the theory, that is, that the equations provable in this theory are precisely the equations true in the category of sets. On the other hand the rewriting behavior of the equations is problematic: as described in [9], confluence fails for the known presentations, and there cannot exist a left-linear confluent presentation.

When the equation $[\lambda\iota{:}\sigma_1.\Delta(\mathsf{in}_1\iota)\,,\ \lambda\iota{:}\sigma_2.\Delta(\mathsf{in}_2\iota)] = \Delta$ is dropped, yielding the theory of cartesian closed categories with *weak sums*, the theory admits a strongly normalizing and confluent presentation, so the resulting theory is decidable. In fact the rewriting theory of the cartesian closed categories with weak sums was the object of intense research activity in the 1990's: a selection of relevant papers might include [6, 1, 9, 5, 12, 7].

So there are rich and well-behaved rewriting theories capturing fragments of $\cong$. But if we want to embrace $\cong$ in its entirety we need to work harder. Ghani [11] presented a complex proof of decidability of the theory via an analysis of a non-confluent rewriting system. The most successful analysis of the theory to date is that by Altenkirch, Dybjer, Hofmann, and Scott [2] based on the semantical technique of Normalization by Evaluation. In that paper a computable function nf is defined mapping terms to "normal forms," together with a computable function d mapping normal forms to terms, satisfying the following theorem.

Theorem 6 ([2])**.** *For every* Δ, $\Delta \cong \mathsf{d}(\mathsf{nf}(\Delta))$*, and for every* Δ_1 *and* Δ_2, $\Delta_1 \cong \Delta_2$ *if and only if* $\mathsf{d}(\mathsf{nf}(\Delta_1))$ *and* $\mathsf{d}(\mathsf{nf}(\Delta_2))$ *are identical.*

Corollary 2. *Equality between type-derivations is decidable.*

6.1 Type theories

It is traditional in treatments of Curry-style typing to consider types modulo a subtyping relation $\leq$ under which the set of types makes a partially ordered set. Following [3] we refer to such a set of inequalities as a *type theory*. It is fairly straightforward to incorporate theories of subtyping into our Church-style system; we outline the development briefly here. It is best to start with the proof-terms, as the carriers of semantic information. As suggested in [18, 24] we need to view subtyping semantically not as simple set-inclusion but as a relationship witnessed by coercion functions. So in the syntax of proof-terms we postulate a signature Σ of names for primitive coercion functions $\mathsf{c} : \sigma \to \tau$. Fixing a signature Σ of coercion constants corresponds to a type theory in the sense of [3] in an intuitively obvious way: each $\mathsf{c} : \sigma \to \tau$ corresponds to an axiom $\sigma \leq \tau$ in the theory.

Conversely, certain type theories can be naturally captured by defining an appropriate signature. The *minimal type theory* Θ from [3] will correspond to coercions defined by the proof-terms as in Figure 2 (without the subtyping rule). That is, Θ, corresponds to the empty signature Σ.

Another important type theory is the theory Π obtained from Θ by adding equations making the lattice of types distributive and adding an "Extended Disjunction Property" axiom

$$\sigma \to (\rho \vee \tau) \leq (\sigma \to \rho) \vee (\sigma \to \tau) \qquad \text{when } \phi \text{ is a Harrop type.}$$

(A type is a Harrop type if the disjunction constructor $\vee$ occurs only in negative position.) The importance of the type theory Π in the Curry-style setting is that under Π Subject Reduction holds for ordinary β-reduction: if $B \vdash M : \sigma$ and either $M =_\beta N$ or $M \to_\eta N$ then $B \vdash N : \sigma$.

We now describe how to construct a signature Σ_Π corresponding to the type theory Π. Recall that this theory is obtained from Θ by adding axioms for distributivity and for the extended disjunction property. The latter rule can be captured by a family of constants

$$\mathsf{dp} : (\sigma \to (\rho \vee \tau)) \to ((\sigma \to \rho) \vee (\sigma \to \tau))$$

We need not add constants capturing the distributivity axiom, since the semantics of our proof-terms is based on categories that are cartesian closed with binary coproducts, and in such categories products always distribute over coproducts. Now to introduce coercions into our Church-style typing system we add the following rule

$$\frac{\Gamma \vdash M@\Delta : \sigma_1 \quad \mathsf{c} : \sigma_1 \to \sigma_2 \in \Sigma}{\Gamma \vdash M@\mathsf{c}\Delta : \sigma_2} \ (\mathit{Coerce})$$

Concerning equality between type-derivations, if we want to reason about equality between type-derivations under the type theory Π we need to take into account the behavior of the basic coercion functions $\mathsf{dp} : (\sigma \to (\rho \vee \tau)) \to ((\sigma \to \rho) \vee (\sigma \to \tau))$.

Whenever the coercions dp are injective we have that for two proof-terms Δ_1 and Δ_2, $\mathsf{dp}\Delta_1 = \mathsf{dp}\Delta_2$ only if $\Delta_1 = \Delta_2$. So in reasoning about such coercions syntactically, there are no additional axioms or rules of inference that apply, in other words *we can treat the* dp *constants as free variables.* Since the techniques of [2] apply perfectly well to open terms, we conclude the following.

Corollary 3. *Equality between type-derivations under the type theory* Π *is decidable.*

7 Future Work

The reduction semantics of our calculus is complex, due to the well-known awkwardness of the $(\vee E)$ rule. Since this is largely due to the global nature of the substitution in the conclusion; this suggests that an explicit substitutions calculus might be better-behaved.

There is a wealth of research yet to be done exploring coherence in the presence of union types: as we have seen the structure of the category of types affects the semantics of derivations. For instance, decidability of equality when coercions are not assumed to be injective needs attention.

We took a fairly naive approach to the semantics of type-derivations in this paper; we were content to derive some results that assumed nothing more about the semantics than cartesian closure and coproducts. But the failure of coherence, implying that the meanings of type-derivations are not "generic," suggests that there is interesting structure to be explored in the semantics of coercions in the presence of unions.

Acknowledgements. We are grateful to Mariangiola Dezani-Ciancaglini for several illuminating discussions about this work.

References

[1] Y. Akama. On Mints' reduction for ccc-calculus. In *TLCA*, volume 664 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1993.

[2] T. Altenkirch, P. Dybjer, M. Hofmann, and P. J. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *LICS*, pages 303–310, 2001.

[3] F. Barbanera, M. Dezani-Ciancaglini, and U. De'Liguoro. Intersection and union types: syntax and semantics. *Inf. Comput.*, 119(2):202–230, 1995.

[4] B. Capitani, M. Loreti, and B. Venneri. Hyperformulae, Parallel Deductions and Intersection Types. *BOTH, Electr. Notes Theor. Comput. Sci.*, 50(2):180–198, 2001.

[5] R. Di Cosmo and D. Kesner. A confluent reduction for the extensional typed lambda-calculus with pairs, sums, recursion and terminal object. In *ICALP*, pages 645–656, 1993.

[6] D. Cubric. Embedding of a free cartesian closed category into the category of sets. Manuscript, McGill University, 1992.

[7] P.-L. Curien and R. Di Cosmo. A confluent reduction for the lambda-calculus with surjective pairing and terminal object. *J. Funct. Program.*, 6(2):299–327, 1996.

[8] P.-L. Curien and G. Ghelli. Coherence of subsumption, minimum typing and type-checking in $F_{<=}$. *Mathematical Structures in Computer Science*, 2(1):55–91, 1992.

[9] D. J. Dougherty. Some lambda calculi with categorical sums and products. In *RTA*, volume 690 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, 1993.

[10] D. J. Dougherty and R. Subrahmanyam. Equality between functionals in the presence of coproducts. *Information and Computation*, 157(1,2):52–83, 2000.

[11] N. Ghani. ßn-equality for coproducts. In *TLCA*, volume 902 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 1995.

[12] C. B. Jay and N. Ghani. The virtues of eta-expansion. *J. Funct. Program.*, 5(2):135–154, 1995.

[13] Donald E. Knuth. Examples of formal semantics. In E. Engeler, editor, *Symposium on Semantics and Algorithmic Languages*, number 188 in LNM, pages 212–235. Springer, 1970.

[14] J. Lambek and P. Scott. *Introduction to Higher-order Categorical Logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1986.

[15] L. Liquori and S. Ronchi Della Rocca. Intersection typed system *à la* Church. *Information and Computation*, 9(205):1371–1386, 2007.

[16] D. MacQueen, G. Plotkin, and R. Sethi. An ideal model for recursive polymorphic types. *Information and Control*, 71:95–130, 1986.

[17] C. Pierce, B. and N. Turner, D. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–247, 1994.

[18] J. C. Reynolds. Using category theory to design implicit conversions and generic operators. In *Semantics-Directed Compiler Generation*, volume 94 of *Lecture Notes in Computer Science*, pages 211–258. Springer, 1980.

[19] J. C. Reynolds. The coherence of languages with intersection types. In *TACS*, volume 526 of *Lecture Notes in Computer Science*, pages 675–700. Springer, 1991.

[20] J. C. Reynolds. Design of the programming language Forsythe. Report CMU–CS–96–146, Carnegie Mellon University, Pittsburgh, Pennsylvania, June 28, 1996.

[21] J. C. Reynolds. *Theories of Programming Languages*. Cambridge University Press, 1998.

[22] S. Ronchi Della Rocca. Intersection typed lambda-calculus. *Electr. Notes Theor. Comput. Sci.*, 70(1), 2002.

[23] S. Ronchi Della Rocca and L. Roversi. Intersection logic. In *CSL*, volume 2142 of *Lecture Notes in Computer Science*, pages 421–428. Springer-Verlag, 2001.

[24] V. Tannen, T. Coquand, C. A. Gunter, and A. Scedrov. Inheritance as implicit coercion. *Inf. Comput.*, 93(1):172–221, 1991.

[25] S. van Bakel, L. Liquori, S. Ronchi della Rocca, and P. Urzyczyn. Comparing Cubes of Typed and Type Assignment System. *Annal of Pure and Applied Logics*, 86(3):267–303, 1997.

[26] J. B. Wells and C. Haack. Branching types. In *ESOP*, volume 2305 of *Lecture Notes in Computer Science*, pages 115–132. Springer-Verlag, 2002.