A Realizability Interpretation for Intersection and Union Types

Daniel J. Dougherty, Ugo de 'Liguoro, Luigi Liquori, Claude Stolze

A Proof-Functional Logic Inspired by Set Types

Daniel J. Dougherty[1], Ugo de'Liguoro[2], Luigi Liquori[3], and Claude Stolze[4]

1 **Worcester Polytechnic Institute, USA**
2 **Università di Torino, Italy**
3 **INRIA Sophia Antipolis-Méditerranée, France**
4 **ENS Rennes and UPMC, France**

Abstract

Proof-functional logical connectives allow reasoning about the structure of logical proofs, so *de facto* giving to the latter the status of *first-class* objects. This is in contrast to classical *truth-functional* connectives where the meaning of a compound formula is dependent only on the truth value of its subformulas. But also with intuitionistic connectives, where only the existence of certain constructions matters, and not their actual shape.

Typed lambda-calculi can serve as formalisms to study proofs as objects and to explore the semantics of logical connectives. At the same time, lambda-calculi with intersection and union types have been studied as tools for analyzing the reduction behavior of terms. The connection between intersection and union types with logics is less clear. Sometimes they are considered the counterpart of proof-functional logical operators, sometimes they are injected (compiled) into well-known logics, such as Combinatory Logic. As such, there is no consensus about *which* lambda-calculus should be taken as a proof-language for logics corresponding to intersection and union type assignment.

In this paper we present a typed lambda calculus, enriched with products, coproducts, explicit coercions, and a related proof-functional logics. This calculus, directly derived by the typed calculus of [7], has been proved isomorphic to the Barbanera-Dezani-de'Liguoro type assignment system. We also present a logic $\mathcal{L}^{\cap\cup}$ featuring two proof-functional connectives, namely strong conjunction and strong disjunction. The logics is suitable to be extended with *relevant implication*, inspired by the B^+ Relevant Logic of Meyer-Routley. We prove the typed calculus to be isomorphic to the logics $\mathcal{L}^{\cap\cup}$ and we give a realizability semantics using *Mints' realizers* and a completeness theorem. A prototype implementation is referred.

1998 ACM Subject Classification Lambda-calculus, Type theory, Logic and verification, Proof theory, Constructive mathematics.

Keywords and phrases Intersection and union types, proof-functional logics, "Church-style" explicitely typed calculi vs. "Curry-style" implicitely typed calculi.

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

This paper is a contribution to the study of intersection and union type systems and their role in *logical* investigations.

There are two well-known points of view on type systems: (i) types as specifications and terms as programs, and (ii) types as propositions and terms as evidence. Let us call the former the "computational" perspective, and the latter the "logical" one.

In the logical view a type judgment $t : \sigma$ is taken to mean that t is a construction providing evidence of the proposition σ, reducing to a canonical element of σ. In this way

typed lambda-calculi are at the core of proof assistants and logical frameworks. In the computational view a typing judgment $t : \sigma$ is taken to mean that t denotes an element of the datatype σ, which may in fact be defined in a way external to the system for making type-judgments.

Within the computational tradition itself there are two approaches: explicitly-typed calculi ("Church-style"), and type assignment systems ("Curry-style"). These represent more than a difference in presentation: in type assignment systems types provide a means for making assertions about the semantics of raw terms, while in explicitly typed calculi types are a method of insuring that only well-behaved terms are considered at all.

The logical view resides naturally in a system of Church-style explicit typing. Existing logical frameworks and proof assistants take such explicitly-typed calculi for their foundation.

Intersection types originated [19, 13, 8] within the computational perspective as a tool for analyzing the functional behavior of lambda-terms: intersection type systems give characterizations of each of the sets of strongly normalizing, weakly normalizing, and head-normalizing terms. From a programming-languages perspective, intersection types support (finitary) *overloading*. Subtyping arises naturally in the study of intersection types.

Later, union types were introduced, as a foundational study [2] and also from programming-languages motivation [14, 5]. Union types are somewhat similar to sum types, but as Pierce [17] notes: *"The main formal diference between disjoint and non-disjoint union types is that the latter lack any kind of* case *construct: if we know only that a value v has type $T_1 \cup T_2$ then the only operations we can safely perform on v are ones that make sense for* both T_1 *and* T_2*".*

Naturally, the question arose whether intersection, union, and subtyping can be given a logical explanation. Pottinger [19] already identified this question: *"Since the meaning of $\cap$ is reasonably clear (to claim that $A \cap B$ is to claim that one has a reason for asserting A which is also a reason for asserting B), it would obviously be of interest to figure out how to add $\cap$ to intuitionist logic and then consider the analysis of intuitionist mathematical reasoning in the light of the resulting system".* A natural logical analogue of computational interpretation of union types is "if we want to reason from an assumption v that $T_1 \cup T_2$ holds, then we may reason separately assuming v is evidence of T_1 and that v is evidence of T_2 as long as we use that evidence *in the same way.*"

There has subsequently been a lot of work on this question of understanding "proof-functional" connectives [12, 16, 3, 1, 15, 6] where the logical analogue of intersection has come to be called "strong conjunction", with "strong disjunction" corresponding to union of course, and, in [6] with subtyping associated with relevant implication, long of interest to philosophers. It became clear that a focus on *realizability* was most fruitful, typically taking untyped terms (from lambda-calculus or combinatory logic) as realizers.

Independent of this thread of research, the question arose whether naturally intersection and union type systems could be presented in Church-style, *i.e.* explicitly typed. There are technical obstacles to an explicitly-typed treatment that would inherit the core properties of the type-assignment approach: subject reduction, subject expansion, strong normalization, unicity of typing, decidability of type reconstruction and type checking. Several proposals [18, 9, 4, 22, 23, 24] were explored, none of which met all the criteria above. The system presented here derives from the system of $\Lambda^{\cap}_{\mathsf{t}}$ [11] subsequently generalized in the system $\Lambda^{\cap\cup}_{\mathsf{t}}$ [7] to include union types. The latter works did not include subtyping, and left open the question of a logical interpretation of the lambda-calculus presented.

All of the work on understanding the logical aspects of intersection, union, and subtyping took place in the Curry-style framework. This was natural given the fact that type assignment

was the most natural framework for intersection and union types, because the typing rules are not *syntax directed.* But the fact that most uses of lambda-calculi in logical systems use explicitly-typed terms poses a compelling question, the main topic of the current paper:

> *Can a logical investigation of intersection and union types, with/out subtyping, take place in the context of an explicitly-typed lambda calculus?*

The motivation is that success here should point the way towards applications of intersection and union types in proof assistants and logical frameworks. The hope is that they can provide as much insight into logical systems as they have in the computational arena.

Contributions

After recalling, in Section 2, the original type assignment system $\Lambda_{\mathsf{u}}^{\cap\cup}$ of [2] and the typed λ-calculus $\Lambda_{\mathsf{t}}^{\cap\cup}$ of [7], we define a new notion, the *essence* of a typed term, central to the technical development that follows. We also present a new proof-functional logic $\mathcal{L}^{\cap\cup}$. Section 3 provides a logical interpretation of $\Lambda_{\mathsf{t}}^{\cap\cup}$ using realizability. We show that the system defined in [7] serves as a bridge between the intersection and union type assignment system and the logic $\mathcal{L}^{\cap\cup}$ based on proof essence. We prove completeness, namely that $\Gamma \vdash M@\Delta : \sigma$ if and only if Δ realizes $G_\Gamma \vdash r_\sigma[M]$ where M is related to Δ by the essence mapping, if and only if $\Gamma \vdash \Delta : \sigma$ in the proof-functional logic.

Section 4 presents further theoretical and pragmatic developments. Subsection 4.1 extends the typed system and the logics by adding a natural notion of subtyping. This is represented in the type assignment system as a non-syntax-directed substitution rule, in the typed calculus as an explicit coercion, and in the logic calculus as another well-known proof-functional connective called *relevant implication.* In Subsection 4.2 we briefly describes our prototype implementation of the type checking and proof inhabitation for the system with intersection/strong conjunction and union/strong disjunction and coercions as relevant implication.

1.1 Related Work

There are far too many studies of type systems featuring intersection, union, and subtyping to identify individually here. We have tried to outline the main currents of research in the introduction; here we will mention some work that is directly related to the contributions of this paper.

The formal investigation of soundness and completeness for a notion of realizability was initiated by Lopez-Escobar [12] and subsequently refined by Mints [16]. It is Mints' approach that we build on here.

The connection between intersection types and relevant implication was noticed by Alessi and Barbanera in [1]. Barbanera and Martini [3] studied three proof-functional *strong conjunction*, the *relevant implication* (see Meyer-Routley's [15] system B^+), and the *strong equivalence* connective for double implication, relating those connectives with suitable type assignments system, a realizability semantics and a completeness theorem.

Dezani-Ciancaglini, Ghilezan, and Venneri [6], investigated a *Curry-Howard* interpretation of intersection and union types (for Combinatory Logic). Using the well understood relation between *combinatory logics* and lambda-calculus, they encode type-free lambda terms in suitable combinatoric logic formulas and then type them using intersection and union types. As they put it, their goal is *"... to set out a logical system ... such that the intersection and union type constructors are interpreted as propositional connectives and then their derivability*

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

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

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

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

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

Figure 1 The Intersection and Union Type Assignment System $\Lambda^{\cap\cup}_{\mathsf{u}}$ [2].

is completely represented by derivability in a logical Hilbert-style, axiomatization." This is a complementary approach to the realizability-based one here.

Barbanera, Dezani, and de'Liguoro [2] presented a untyped lambda-calculus with related type assignment system featuring intersection and union types. Our previous work [7] presented a typed calculus that explored the relationship between the proof-functional intersections and unions and the truth-functional products and sums; the intersection and union aspect of the system was isomorphic, after erasure, to the Barbanera-Dezani-de'Liguoro [2] type assignment system.

2 The Typed Lambda Calculus $\Lambda^{\cap\cup}_{\mathsf{t}}$

The type assignment system $\Lambda^{\cap\cup}_{\mathsf{u}}$ is the set of inference rules for assigning intersection and union types to terms of the pure lambda-calculus. The inference rules are presented in Figure 1. In [7] we introduced a typed lambda-calculus $\Lambda^{\cap\cup}_{\mathsf{t}}$ whose goal was to capture a decidable and Church-style version of the Curry-style $\Lambda^{\cap\cup}_{\mathsf{u}}$. The pseudo-terms of the $\Lambda^{\cap\cup}_{\mathsf{t}}$ calculus have the form $M@\Delta$, where M and Δ have the following syntax:

$$\begin{array}{rcl} M & ::= & x_\iota \mid \lambda x_\iota.M \mid M\,M \\ \Delta & ::= & \iota \mid * \mid \lambda\iota{:}\sigma.\Delta \mid \Delta\,\Delta \mid \langle \Delta\,,\,\Delta\rangle \mid [\Delta\,,\,\Delta] \mid \mathsf{pr}_i\Delta \mid \mathsf{in}_i\Delta \qquad i = 1,2 \end{array}$$

The typed judgments are of the shape $\Gamma^{@} \vdash M@\Delta : \sigma$, where in a nutshell M is a type-free lambda-term, Δ is a typed lambda-term enriched with product, coproducts, projections, and injections to faithfully "memorize" every step of a type assignment derivation, and $\Gamma^{@}$ contains declarations of the shape $x_\iota@\iota{:}\sigma$, where x_ι and ι are free-variables of M and Δ, respectively. The inference rules are presented in Figure 2. The main feature of the system was to keep M to be "synchronized" with Δ. As example, we can derive the judgement $\vdash \lambda x_\iota.x_\iota@\langle\lambda\iota{:}\sigma_1.\iota\,,\,\lambda\iota{:}\sigma_2.\iota\rangle : (\sigma_1 \to \sigma_1) \cap (\sigma_2 \to \sigma_2)$. A term $[\Delta_1\,,\,\Delta_2]$ corresponds to the pairing of two arrows Δ_i ($i = 1,2$) to build an arrow out of a coproduct type. In fact, 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

$\Gamma^{@} \triangleq \{x_{\iota_1}@\iota_1{:}\sigma_1, \ldots, x_{\iota_n}@\iota_n{:}\sigma_n\}$ $\iota_i \neq \iota_j$ implies $x_{\iota_i} \neq x_{\iota_j}$, and $\Gamma^{@}, x_\iota@\iota{:}\sigma \triangleq \Gamma^{@} \cup \{x_\iota@\iota{:}\sigma\}$

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

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

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

$$\frac{\Gamma^{@} \vdash M@\Delta : \sigma_i \quad i \in \{1,2\}}{\Gamma^{@} \vdash M@\mathsf{in_i}\Delta : \sigma_1 \cup \sigma_2}\ (\cup I_i)$$

$$\frac{\Gamma^{@}, x_\iota@\iota{:}\sigma_1 \vdash M@\Delta_1 : \sigma_3 \quad \Gamma^{@}, x_\iota@\iota{:}\sigma_2 \vdash M@\Delta_2 : \sigma_3 \quad \Gamma^{@} \vdash N@\Delta_3 : \sigma_1 \cup \sigma_2}{\Gamma^{@} \vdash M\{N/x_\iota\}@[\lambda\iota{:}\sigma_1.\Delta_1\,,\,\lambda\iota{:}\sigma_2.\Delta_2] \cdot \Delta_3 : \sigma_3}\ (\cup E)$$

Figure 2 The Typed Calculus $\Lambda_{\mathsf{t}}^{\cap\cup}$ [7].

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). As is typical we name the unique element of the terminal object as $*$.

The relation between untyped and typed reductions is subtle because of the presence of the "Gross-Knuth" parallel reduction in the untyped calculus and a fairly complex notion of synchronization of M and Δ, via synchronized β- and Δ-reductions in the typed calculus. In a nutshell, 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 ($\cap I$), ($\cup I$) or ($\cap E$) rules, which change the Δ but not the M. This disconnection is "logged" in the Δ via occurrences of operators $\langle -\,,\,-\rangle$, $[-\,,\,-]$, $\mathsf{pr_i}$, and $\mathsf{in_i}$. 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 an *ad hoc* notion of *"overlapping"* that help to define a redex taking into account the surrounding context. Therefore, we define $\Rightarrow$ as the union of two reductions: $\Rightarrow_\beta$ dealing with β-reduction occurring in both M and Δ, and $\Rightarrow_\Delta$ dealing with reductions arising from reduction only in Δ. We refer to the complete reduction definition in [7]. Here are some main properties of the system $\Lambda_{\mathsf{t}}^{\cap\cup}$.

▶ **Theorem 1** (Main properties of $\Lambda_{\mathsf{t}}^{\cap\cup}$ [7]).
Subject reduction. *If $\Gamma \vdash M@\Delta : \sigma$ and $M@\Delta \Rightarrow M'@\Delta'$, then $\Gamma \vdash M'@\Delta' : \sigma$.* ◀
Church-Rosser. *The reduction relation $\Rightarrow$ is confluent.* ◀
Strong normalization. *If $M@\Delta$ is typable without using rule (ω) then M is strongly normalizing.* ◀

Type reconstruction algorithm.

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

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

Type checking algorithm. *There is an algorithm* Typecheck *satisfying:* $\Gamma \vdash M@\Delta : \sigma$ *if and only if* $\mathsf{Typecheck}(\Gamma, M@\Delta, \sigma) = \mathsf{true}$*.*

Judgment decidability. *It is decidable whether* $\Gamma \vdash M@\Delta : \sigma$ *is derivable.* ◀

Isomorphism of typed-untyped derivations. *Let* $\mathcal{D}er\Lambda^{\cap\cup}_{\mathsf{u}}$ *and* $\mathcal{D}er\Lambda^{\cap\cup}_{\mathsf{t}}$ *be the sets of all (un)typed derivations, let* $\mathcal{D}_{\mathsf{u}}$*, and* $\mathcal{D}_{\mathsf{t}}$ *denote (un)typed derivations, respectively, and let define the functions* $\mathcal{F} : \mathcal{D}er\Lambda^{\cap\cup}_{\mathsf{t}} \Rightarrow \mathcal{D}er\Lambda^{\cap\cup}_{\mathsf{u}}$ *and* $\mathcal{G} : \mathcal{D}er\Lambda^{\cap\cup}_{\mathsf{u}} \Rightarrow \mathcal{D}er\Lambda^{\cap\cup}_{\mathsf{t}}$*. The systems* $\Lambda^{\cap\cup}_{\mathsf{t}}$ *and* $\Lambda^{\cap\cup}_{\mathsf{u}}$ *are isomorphic in the following sense:* $\mathcal{F} \circ \mathcal{G}$ *is the identity in* $\mathcal{D}er\Lambda^{\cap\cup}_{\mathsf{u}}$ *and* $\mathcal{G} \circ \mathcal{F}$ *is the identity in* $\mathcal{D}er\Lambda^{\cap\cup}_{\mathsf{t}}$ *modulo uniform naming of variable-marks, where* $\mathcal{F}$ *and* $\mathcal{G}$ *are functions i.e.,* $\mathcal{G}(\mathcal{F}(\Gamma \vdash M@\Delta : \sigma)) = \mathsf{ren}(\Gamma) \vdash \mathsf{ren}(M@\Delta) : \sigma$*, where* ren *is a simple function renaming the free occurrences of variable-marks.* ◀

2.1 The proof essence partial function

We start with a simple question: assuming $M@\Delta$ is derivable, can we *extract* the computational part M from a proof-term Δ? Luckily the answer is positive. To do that, let us extend the pure lambda-calculus syntax by a constant Ω, typable by ω only, and consider the least pre-order including α-conversion, η-conversion, and the compatible closure of $\Omega \sqsubseteq M$ for any M. Then $\sqsubseteq$ is a partial order and the set of extended λ-terms is closed under sups of compatible terms: M and N are compatible, usually written $M \uparrow N$, if $M \sqsubseteq P \sqsupseteq N$ for some P; then the join $M \sqcup N$ exists. Let us define the *essence* of a Δ, written $\wr\,\Delta\,\wr$, as a partial mapping as follows:

▶ **Definition 2** (Proof essence)**.** The type-free *essence* M of a typed proof Δ is:

$$\begin{array}{rcl@{\qquad}rcl}
\wr * \wr & \triangleq & \Omega & \wr \iota \wr & \triangleq & x_\iota \\
\wr \lambda\iota{:}\sigma_1.\Delta \wr & \triangleq & \lambda x_\iota.\wr\,\Delta\,\wr & \wr\,\Delta_1\,\Delta_2\,\wr & \triangleq & \wr\,\Delta_1\,\wr\,\wr\,\Delta_2\,\wr \\
\wr\,[\lambda\iota{:}\sigma_1.\Delta_1\,,\,\lambda\iota{:}\sigma_2.\Delta_2]\cdot\Delta_3\,\wr & \triangleq & (\wr\,\Delta_1\,\wr \sqcup \wr\,\Delta_2\,\wr)\{\wr\,\Delta_3\,\wr/x_\iota\} & \wr\,\mathsf{in}_{\mathsf{i}}\Delta\,\wr & \triangleq & \wr\,\Delta\,\wr \\
\wr\,\langle\Delta_1\,,\,\Delta_2\rangle\,\wr & \triangleq & \wr\,\Delta_1\,\wr \sqcup \wr\,\Delta_2\,\wr & \wr\,\mathsf{pr}_{\mathsf{i}}\Delta\,\wr & \triangleq & \wr\,\Delta\,\wr
\end{array}$$

Intuitively, the intended meaning of the *"essence"* partial map is that it takes a typed proof-term Δ in the typed calculus $\Lambda^{\cap\cup}_{\mathsf{t}}$ of [7] and produces a type-free lambda-term M that to be used as a *Mints' realizer* of a type σ considered as a logical formula. Note that M and Δ are both also typable with σ using the type assignment and the type system, respectively. Summarizing, the signature of the essence is as follows:

$$\wr\,-\,\wr : \text{proof-terms } (\Delta\text{'s}) \to \text{untyped lambda-terms } (M\text{'s}).$$

2.2 The Proof-functional Logic $\mathcal{L}^{\cap\cup}$

Indeed, for a given typable Δ, the left-hand side of the @, namely M, can be omitted since it represents just the essence of Δ, *i.e.* $\wr\,\Delta\,\wr \sqsubseteq M$. Thus we can introduce the proof-functional logic, called $\mathcal{L}^{\cap\cup}$ and presented in Figure 3. The following theorems hold:

▶ **Theorem 3.** *If* $\Gamma^{@} \vdash M@\Delta : \sigma$ *then* $\wr\,\Delta\,\wr$ *is defined and* $\wr\,\Delta\,\wr \sqsubseteq M$*.* ◀

▶ **Theorem 4** (Equivalence)**.** *Let* Γ *be obtained by* $\Gamma^{@}$*, simply by erasing all the "*$x@$*". Then* $\Gamma^{@} \vdash M@\Delta : \sigma$ *if and only if* $\Gamma \vdash \Delta : \sigma$ *and* $\wr\,\Delta\,\wr \sqsubseteq M$*.* ◀

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

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

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

$$\frac{\Gamma \vdash \Delta_1 : \sigma_1 \quad \Gamma \vdash \Delta_2 : \sigma_2 \quad \wr \Delta_1 \wr \uparrow \wr \Delta_2 \wr}{\Gamma \vdash \langle \Delta_1\, , \Delta_2 \rangle : \sigma_1 \cap \sigma_2}\ (\cap I) \qquad \frac{\Gamma \vdash \Delta : \sigma_1 \cap \sigma_2 \quad i \in \{1,2\}}{\Gamma \vdash \mathsf{pr}_i\Delta : \sigma_i}\ (\cap E_i)$$

$$\frac{\Gamma \vdash \Delta : \sigma_i \quad i \in \{1,2\}}{\Gamma \vdash \mathsf{in}_i\Delta : \sigma_1 \cup \sigma_2}\ (\cup I_i) \qquad \frac{\begin{array}{c}\Gamma, \iota{:}\sigma_1 \vdash \Delta_1 : \sigma_3 \quad \wr \Delta_1 \wr \uparrow \wr \Delta_2 \wr \\ \Gamma, \iota{:}\sigma_2 \vdash \Delta_2 : \sigma_3 \quad \Gamma \vdash \Delta_3 : \sigma_1 \cup \sigma_2\end{array}}{\Gamma \vdash [\lambda\iota{:}\sigma_1.\Delta_1\, , \lambda\iota{:}\sigma_2.\Delta_2] \cdot \Delta_3 : \sigma_3}\ (\cup E)$$

Figure 3 The proof-functional logic $\mathcal{L}^{\cap\cup}$.

Since $\mathcal{L}^{\cap\cup}$ is a proof-functional logic it is natural to consider the pair "$\Delta : \sigma$" as a logical formula. Pictorially speaking, we could say that the type assignment system of [2] and the logic $\mathcal{L}^{\cap\cup}$ are "bridged" by the typed system $\Lambda_{\mathsf{t}}^{\cap\cup}$, and the above Theorems prove this fact: the key new concept is the essence partial map. This is, to the best of our knowledge, the first attempt to interpret union as a proof-functional connective.

3 Realizability interpretation of $\Lambda_{\mathsf{t}}^{\cap\cup}$

In this section we will show that "essence" is a syntactical transformation relating two distinct systems, intuitionism and a proof-functional logic.

From Theorem 4 we know that if $\Gamma \vdash M@\Delta : \sigma$ then there is a tight relation among Δ and M, which is captured by the essence mapping. Comparing system $\Lambda_{\mathsf{t}}^{\cap\cup}$ to the original $\Lambda_{\mathsf{u}}^{\cap\cup}$ it is easily seen that Δ is a proof-term of the statement $M : \sigma$ in system $\Lambda_{\mathsf{u}}^{\cap\cup}$; but Δ is a simply typed term: in fact if we drop the restriction concerning the "essence" in rules $(\cap I)$ and $(\cup E)$ in system $\mathcal{L}^{\cap\cup}$ replacing $\sigma \cap \tau$ by $\sigma \times \tau$ and $\sigma \cup \tau$ by $\sigma + \tau$ then we get a simply typed lambda-calculus with product and sums, namely the intuitionistic propositional logic with implication, conjunction and disjunction in disguise.

We will provide a foundation for the proof-functional logic $\mathcal{L}^{\cap\cup}$ by interpreting the $\mathcal{L}^{\cap\cup}$ into an extension of Mints' provable realizability. However when proving a formula $r_\sigma[M]$ we have two kinds of realizers: the former is the untyped lambda term M, that we propose to call just a "method" borrowing terminology from Barbanera-Martini. The second kind are Δ's that turn out to be realizers in the ordinary sense of Mints' calculus, when it is presented as a particular intuitionistic theory.

Therefore, we prove a completeness proof that this is the case, namely that $\Gamma \vdash \Delta : \sigma$ if and only if Δ realizes $G_\Gamma \vdash r_\sigma[M]$ for some M related to Δ by the essence mapping, if and only if $\Gamma \vdash \Delta : \sigma$ is derivable in $\mathcal{L}^{\cap\cup}$.

For this aim we use and extend Mints' approach of Provable Realizability [16, 1, 3]. We interpret the statement $\vdash M@\Delta : \sigma$ as "Δ is a construction of $M : \sigma$"; on the other hand

$M : \sigma$ is the meaning of the formula $r_\sigma[M]$, provided that we extend the notion to cope with union types; the latter formula reads as "M is a method to assess σ" in terms of [12, 3]; now the meaning of Δ is that of a constructive proof of $r_\sigma[M]$, and hence it is a "realizer" of this formula. In short we have two kind of realizers on two levels: the M, which is a Mints' realizer of σ, and the Δ which is an ordinary realizer, in the sense of standard BHK interpretation of intuitionistic logic, of the statement $r_\sigma[M]$.

To avoid confusion, in the following we shall reserve the word "realizer" for the Δ-terms, and we will use the word "method" referring to the untyped λ-term M.

▶ **Definition 5.** Let $\mathbf{P}_\phi(x)$ be a unary predicate for each atomic type ϕ. Then we define the predicates $r_\sigma[x]$ for each type σ by induction over σ, as the first order logical formulae:

$$\begin{aligned} r_\phi[x] &\equiv \mathbf{P}_\phi(x) \\ r_{\sigma_1\to\sigma_2}[x] &\equiv \forall y.r_{\sigma_1}[y] \supset r_{\sigma_2}[x\,y] \\ r_{\sigma_1\cap\sigma_2}[x] &\equiv r_{\sigma_1}[x] \wedge r_{\sigma_2}[x] \\ r_{\sigma_1\cup\sigma_2}[x] &\equiv r_{\sigma_1}[x] \vee r_{\sigma_2}[x] \end{aligned}$$

In the above $\supset$, $\wedge$ and $\vee$ are the logical connectives for implication, conjunction and disjunction respectively, that must be kept distinct from $\cap$ and $\cup$. In the first order language whose terms are type-free lambda-terms, we have formulas of the shape $r_\sigma[M]$, whose intended meaning is that M is a method for σ in the intersection-union type discipline. Note that in $r_\sigma[x]$ the term-variable x is the only free-variable; in particular in $r_{\sigma_1\to\sigma_2}[M] \equiv \forall y.r_{\sigma_1}[y] \supset r_{\sigma_2}[M\,y]$ we assume that $y \notin \mathsf{Fv}(M)$.

By NJ we mean the natural-deduction presentation of the intuitionistic first-order predicate calculus. Derivations in NJ are trees of judgments $G \vdash A$, where G is the set of undischarged assumptions, rather than trees of formulas as in Gentzen's original formulation.

▶ **Definition 6** (The system $\mathsf{NJ}(\beta)$)**.** The system $\mathsf{NJ}(\beta)$ is the natural deduction system for first order intuitionistic logic with untyped λ-terms and predicates $\mathbf{P}_\phi(x)$, the latter being axiomatized via the Post rules:

$$\frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \mathbf{P}_\phi(M) \quad M =_\beta N}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \mathbf{P}_\phi(N)}\ (Ax\beta) \qquad \frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \mathbf{P}_\phi(M) \quad M =_\eta N}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \mathbf{P}_\phi(N)}\ (Ax\eta) \qquad \frac{}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \mathbf{P}_\omega(M)}\ (Ax\omega)$$

If A is a formula of $\mathsf{NJ}(\beta)$ and $G \triangleq \{A_1, \ldots, A_n\}$ is a set of formulæ (a context), then we write $G \vdash_{\mathsf{NJ}(\beta)} A$ to mean that A is derivable in G.

To the context $\Gamma = \{\iota_1{:}\sigma_1, \ldots, \iota_n{:}\sigma_n\}$ of the logic $\mathcal{L}^{\cap\cup}$ we associate the $\mathsf{NJ}(\beta)$ context $G_\Gamma \triangleq r_{\sigma_1}[x_{\iota_1}], \ldots, r_{\sigma_n}[x_{\iota_n}]$. Note that $G_{\Gamma,\iota:\sigma} = G_\Gamma, r_\sigma[x_\iota]$ and $x_\iota \notin \mathsf{Fv}(G_\Gamma)$, since $\iota \notin Dom(\Gamma)$, by context definition.

▶ **Lemma 7.** *The following rules are admissible in* $\mathsf{NJ}(\beta)$*:*

$$\frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A\{M/x\} \quad M =_\beta N}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A\{N/x\}}\ (Eq\beta) \qquad \frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A\{M/x\} \quad M =_\eta N}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A\{N/x\}}\ (Eq\eta)$$

Proof. By induction over the proof of $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A\{N/x\}$. ◀

▶ **Lemma 8.** *The following rules are admissible in* $\mathsf{NJ}(\beta)$*:*

$$\frac{G_\Gamma, r_{\sigma_1}[x] \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_2}[M]}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\to\sigma_2}[\lambda x.M]} \qquad \frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\to\sigma_2}[M] \quad G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1}[N]}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_2}[M\,N]}$$

$$\frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1}[M] \quad G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_2}[M]}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\cap\sigma_2}[M]} \qquad \frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\cap\sigma_2}[M] \quad i\in\{1,2\}}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_i}[M]}$$

$$\frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_i}[M] \quad i\in\{1,2\}}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\cup\sigma_2}[M]} \qquad \frac{\begin{array}{l} G_\Gamma, r_{\sigma_1}[x] \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_3}[M] \\ G_\Gamma, r_{\sigma_2}[x] \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_3}[M] \quad G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\cup\sigma_2}[N] \end{array}}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_3}[M\{N/x\}]}$$

In spite of the similarity of the rules in Lemma 8 with those of system $\mathcal{L}^{\cap\cup}$ there are no restrictions to the shape of the derivations of the $r_\sigma[M]$ which are needed $\mathcal{L}^{\cap\cup}$ as a system of strong-conjunction $\cap$ and strong-disjunction $\cup$. This is due to the fact the last lemma is about derivations of the predicate $r_\sigma[M]$ and not just of the proof-functional "formula" σ. Nonetheless we have:

▶ **Lemma 9.** *If* $\Gamma \overset{@}{\vdash} M@\Delta : \sigma$ *in system* $\Lambda^{\cap\cup}_{\mathsf{t}}$ *then* $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$.

Proof. By induction over the derivation of $\Gamma \overset{@}{\vdash} M@\Delta : \sigma$ using Lemma 8. ◀

▶ **Theorem 10** (Soundness)**.** *If* $\Gamma \vdash \Delta : \sigma$ *is derivable in* $\mathcal{L}^{\cap\cup}$ *then there exists* M *such that* $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$.

Proof. By Theorem 3 if $\Gamma \vdash \Delta : \sigma$ is derivable then $\Gamma \overset{@}{\vdash} M@\Delta : \sigma$ for some $M \sqsupseteq \wr\,\Delta\,\wr$. The thesis follows by Lemma 9.

◀

We say that the derivation of $G_\Gamma \vdash r_\sigma[M]$ is *standard* if it uses only the rules of the Post system, rules $(Eq\beta)$, $(Eq\eta)$ and the rules from Lemma 8; then we write $G_\Gamma \vdash_S r_\sigma[M]$.

Recall that $\mathsf{NJ}(\beta)$ is a particular case of systems called **I(S)** in [20], which enjoys the property of being strongly normalizable. The normal form, of a derivation, called "fully normal derivation" by Prawitz, is split into a topmost "analytical part" consisting of elimination rules, an intermediate "minimum part" consisting of rules of the Post system, and a final "synthetical part" (ending with the very conclusion of the derivation) only consisting of introduction rules. This implies the subformula property.

▶ **Lemma 11.** *If* $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$ *then* $G_\Gamma \vdash_S r_\sigma[M]$.

Proof. By induction over the fully-normal derivation of $G_\Gamma \vdash r_\sigma[M]$, and then by cases of σ. If σ is ϕ or ω then both the analytic and the synthetic parts are empty, and the thesis is immediate. Otherwise:

Case $\sigma = \sigma_1 \cap \sigma_2$. Since $r_{\sigma_1\cap\sigma_2}[M] \equiv r_{\sigma_1}[M] \wedge r_{\sigma_2}[M]$, the fully-normal derivation of $G_\Gamma \vdash r_{\sigma_1}[M] \wedge r_{\sigma_2}[M]$ must end with $(\wedge I)$, whose premises are $G_\Gamma \vdash r_{\sigma_i}[M]$, $i = 1, 2$ and the thesis follows by induction.

Case $\sigma = \sigma_1 \to \sigma_2$. We have $r_{\sigma_1\to\sigma_2}[M] \equiv \forall y.r_{\sigma_1}[y] \supset r_{\sigma_2}[M\,y]$, so that the synthetic part ends by:

$$\dfrac{\dfrac{G_\Gamma, r_{\sigma_1}[y] \vdash r_{\sigma_2}[M\,y]}{G_\Gamma \vdash r_{\sigma_1}[y] \supset r_{\sigma_2}[M\,y]}\ (\supset I)}{G_\Gamma \vdash \forall y.r_{\sigma_1}[y] \supset r_{\sigma_2}[M\,y]}\ (\forall I)$$

where $y \notin \mathsf{Fv}(G_\Gamma) \cup \mathsf{Fv}(M)$ because of the side condition of rule $(\forall I)$ and the definition of $r_{\sigma_1\to\sigma_2}[M]$. By induction $G_\Gamma, r_{\sigma_1}[y] \vdash_S r_{\sigma_2}[M\,y]$, from which we obtain the standard derivation:

$$\dfrac{\dfrac{G_\Gamma, r_{\sigma_1}[y] \vdash_S r_{\sigma_2}[M\,y]}{G_\Gamma \vdash_S r_{\sigma_1\to\sigma_2}[\lambda y.M\,y]} \qquad \lambda y.M\,y =_\eta M}{G_\Gamma \vdash_S r_{\sigma_1\to\sigma_2}[M]}$$

Case $\sigma_1 \cup \sigma_2$. Then $r_{\sigma_1\cup\sigma_2}[M] \equiv r_{\sigma_1}[M] \vee r_{\sigma_2}[M]$ and the fully-normal derivation of $G_\Gamma \vdash r_{\sigma_1}[M] \vee r_{\sigma_2}[M]$ ends by $(\vee I)$, therefore by induction $G_\Gamma \vdash_S r_{\sigma_i}[M]$ with $i \in \{1,2\}$ and the thesis follows.

◀

▶ **Definition 12** (Δ-realizability)**.** We say that a closed Δ *realizes* the formula $r_\sigma[M]$, written $\Delta \Vdash r_\sigma[M]$, if $\wr\,\Delta\,\wr \sqsubseteq M$ and:

$$\begin{array}{rcl}
\Delta \Vdash r_\phi[M] & & \text{always} \\
\Delta \Vdash r_\omega[M] & \Leftrightarrow & \Delta \equiv * \\
\Delta \Vdash r_{\sigma\to\tau}[M] & \Leftrightarrow & \exists M' =_{\beta\eta} M.\, \forall \Delta', N.\ \Delta' \Vdash r_\sigma[N] \Rightarrow (\Delta\,\Delta') \Vdash r_\tau[M'N] \\
\Delta \Vdash r_{\sigma\cap\tau}[M] & \Leftrightarrow & \Delta \equiv \langle \Delta_1\,,\,\Delta_2\rangle \ \&\ \Delta_1 \Vdash r_\sigma[M] \ \&\ \Delta_2 \Vdash r_\tau[M] \\
\Delta \Vdash r_{\sigma\cup\tau}[M] & \Leftrightarrow & (\Delta \xrightarrow{*} \mathsf{in}_1\Delta_1 \ \&\ \Delta_1 \Vdash r_\sigma[M]) \ \vee\ (\Delta \xrightarrow{*} \mathsf{in}_2\Delta_2 \ \&\ \Delta_2 \Vdash r_\tau[M])
\end{array}$$

We then define $\Delta \Vdash G_\Gamma \vdash r_\sigma[M]$ where Δ is a possibly open term s.t. $\mathsf{Fv}(\Delta) = \{\iota_1, \ldots, \iota_k\} \subseteq \mathsf{Fv}(\Gamma)$, if and only if for all closed $\Delta_1, \ldots, \Delta_k$ and terms $N_1, \ldots, N_k$ such that $\Delta_i \Vdash r_{\Gamma(\iota_i)}[N_i]$ for all $i = 1, \ldots, k$ it is the case that (writing $x_i \equiv x_{\iota_i}$):

$$\Delta\{\Delta_1/\iota_1\}\cdots\{\Delta_k/\iota_k\} \Vdash r_\sigma[M\{N_1/x_1\}\cdots\{N_k/x_k\}].$$

▶ **Lemma 13.** *If $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$ then there exists Δ such that $\Delta \Vdash G_\Gamma \vdash r_\sigma[M]$.*

Proof. By Lemma 11 we can argue by induction over the standard derivation of $G_\Gamma \vdash r_\sigma[M]$. If it ends by a Post rule then the thesis is trivial. Suppose that it ends by the inference

$$\dfrac{G_\Gamma \vdash r_{\sigma_1}[M] \quad G_\Gamma \vdash r_{\sigma_2}[M]}{G_\Gamma \vdash r_{\sigma_1\cap\sigma_2}[M]}$$

Then by induction there are Δ_1, Δ_2 such that $\wr\,\Delta_i\,\wr \sqsubseteq M$ and $\Delta_i \Vdash G_\Gamma \vdash r_{\sigma_i}[M]$. Taking $\Delta \equiv \langle \Delta_1\,,\,\Delta_2\rangle$ we have that $\wr\,\Delta_1\,\wr \sqsubseteq M \sqsupseteq \wr\,\Delta_2\,\wr$ and $\wr\,\Delta\,\wr = \wr\,\Delta_1\,\wr \sqcup \wr\,\Delta_2\,\wr \sqsubseteq M$ hence $\Delta \Vdash G_\Gamma \vdash r_{\sigma_1\cap\sigma_2}[M]$. All other cases are similar. ◀

▶ **Lemma 14.** *If $\Delta \Vdash G_\Gamma \vdash r_\sigma[M]$ then there exists N and Δ' such that $M =_{\beta\eta} N$ and $\Gamma^{@} \vdash N @ \Delta' : \sigma$.*

Proof. By induction over σ. ◀

▶ **Theorem 15** (Completeness)**.** *If $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$ then there exists $N =_{\beta\eta} M$ and Δ such that $\Gamma^{@} \vdash N @ \Delta : \sigma$ and therefore $\Gamma \vdash \Delta : \sigma$.*

Proof. By the hypothesis and Lemma 13 we know that there is a Δ' such that $\Delta' \Vdash G_\Gamma \vdash r_\sigma[M]$. By Lemma 14 this implies that $\Gamma^{@} \vdash N @ \Delta : \sigma$ for some Δ and $N =_{\beta\eta} M$, and we conclude by Theorem 3.

◀

4 Further logical developments and implementation

The results presented here are part of a larger project to build a logical framework, of the LF family, featuring intersection, union types, coercions, and type isomorphism. The following two parts record the next points in either the theoretical and implementation agenda.

4.1 Implicit subtyping as explicit coercions

The logic $\mathcal{L}^{\cap\cup}$ does not encompass the subtyping relation treated in [2], that extends the subtyping relation among intersection types introduced in [8]. Given such a relation, the subsumption rule takes the form:

$$\frac{B \vdash M : \sigma \quad \sigma \leq \tau}{B \vdash M : \tau} \quad (Sub)$$

This rule pairs the intersection and union introduction rules because the subject M of the conclusion the same than in the premise. This calls for a consistent treatment on the side of the Δ's that are typed terms. In [7] it was hinted that the subtyping as coercion should be the proper approach, in the sense that whenever $\sigma \leq \tau$ there should exist a coercion lambda-term $coe_{\sigma\leq\tau} : \sigma \to \tau$ such that the following rule is sound:

$$\frac{\Gamma \vdash \Delta : \sigma \quad \sigma \leq \tau}{\Gamma \vdash (coe_{\sigma\leq\tau}\, \Delta) : \tau} \quad (coe)$$

According to the logic $\mathcal{L}^{\cap\cup}$ this rule is sound if $\wr\, coe_{\sigma\leq\tau}(\Delta) \wr \sqsubseteq M$, while according to the realizability interpretation this is the case if realizers of $r_\sigma[M]$ are sent to realizers of $r_\tau[M]$. We argue that this is the case by showing that, at least for the type theory Ξ from [2], we can establish the following (the proof is omitted because of lack of space):

▶ **Theorem 16.** *If $\sigma \leq \tau \in \Xi$ then there exists a combinator $coe_{\sigma\leq\tau}$ such that $\vdash coe_{\sigma\leq\tau} : \sigma \to \tau$ is a theorem of $\mathcal{L}^{\cap\cup}$ and $\wr\, coe_{\sigma\leq\tau} \wr \sqsubseteq \lambda x.x$.*

We end this section by observing that Theorem 16 is in accordance with the logical interpretation of intersection types proposed in [3]. In fact from the logical point of view subtyping of intersection (and union) types corresponds to inject concepts and rules proper to the *Minimal Relevant Logical* system B^+ introduced by Meyer-Routley in '72. As nicely explained in the Barbanera-Martini paper, the *relevant implication*, denoted by $\supset_r$ from the logic side and $\to_r$ from the type side, capture the behavior of the coercion function $coe_{\sigma\leq\tau}$ as follows:

> "To assert $\sigma \to_r \tau$ (*a.k.a.* $\sigma \leq \tau$) is to assert that any proof-inhabitant of σ *is also* a proof-inhabitant of τ".

Our system then meets the latter requirement because any coercion is "essentially" the identity.

4.2 Prototype Implementation

The actual implementation is part of a larger project including the design and the implementation of an interactive proof assistant, featuring dependent types *à la* Edinburgh Logical Framework, and a variety of proof-functional connectives, such as intersection types (strong conjunction), union types (strong disjunction), type coercions to capture relevant implication,

and type isomorphism (strong equivalence) to capture provable isomorphism in lambda-theories. The prototype is written in the functional language ML. Its *Read-Eval-Print-Loop* (REPL) can read a file containing some signatures, and process it using a lexer, then a parser. Then it can do the following actions, namely (i) type-check the proof, or (ii) normalize the proof using strong reduction, (iii) add some definition in the global context, and (iv) we are currently working on an interactive type inhabitation algorithm.

We implemented exactly the $\Lambda_t^{\cap\cup}$ calculus: we have added a wildcard type called "?" to deal with union introduction, and we added an unification algorithm to apply eliminations rule for implication and union types. The actual type system feature intersection, union and dependent-types, while explicit coercions and strong equivalence are on the top of our stack. The aims of the prototype is to check the expressiveness of the proof-functional nature of the logical engine in the sense that when the user must prove *e.g.* a strong conjunction formula $\sigma_1 \cap \sigma_2$ obtaining (mostly interactively) a witness Δ_1 for σ_1, the prototype could "squeeze" the essence M of Δ_1 to accelerate, and in some case automatize, the construction of a witness Δ_2 proof for the formula σ_2 having the same essence M of Δ_1. The actual state of the prototype can be retrieved on

`https://www.dropbox.com/sh/rd80w81y0lwmw42/AACy0u_PEzXi1w2YJmsmV-wca?dl=0`

References

1 Fabio Alessi and Franco Barbanera. Strong conjunction and intersection types. In *MFCS*, pages 64–73, 1991. URL: `http://dx.doi.org/10.1007/3-540-54345-7_49`, `doi:10.1007/3-540-54345-7_49`.

2 F. Barbanera, M. Dezani-Ciancaglini, and U. de'Liguoro. Intersection and union types: syntax and semantics. *Inf. Comput.*, 119(2):202–230, 1995. `doi:http://dx.doi.org/10.1006/inco.1995.1086`.

3 F. Barbanera and S. Martini. Proof–functional connectives and realizability. *Archive for Mathematical Logic*, 33:189–211, 1994.

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 Mario Coppo and Alberto Ferrari. Type inference, abstract interpretation and strictness analysis. *Theoretical Computer Science*, 121(1):113–143, 1993.

6 M. Dezani-Ciancaglini, S. Ghilezan, and B. Venneri. The "relevance" of intersection and union types. *Notre Dame Journal of Formal Logic*, 38(2):246–269, 1997.

7 Daniel J. Dougherty and Luigi Liquori. Logic and computation in a lambda calculus with intersection and union types. In *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, pages 173–191, 2010.

8 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.

9 Reynolds J.C. Design of the programming language forsythe. In O'Hearn and Tennent ed.s, editors, *Algol-like Languages*. Birkhauser, 1996.

10 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.

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

12 E. G. K. Lopez-Escobar. Proof functional connectives. In *″ ″*, volume 1130 of *Lecture Notes in Mathematics*, pages 208–221. Springer-Verlag, 1985.

13 Coppo M. and Dezani-Ciancaglini M. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Logic*, 21(4):685–693, 1980.

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

15 Robert K Meyer and Richard Routley. Algebraic analysis of entailment I. *Logique et Analyse*, 15:407–428, 1972.

16 Grigori Mints. The completeness of provable realizability. *Notre Dame Journal of Formal Logic*, 30(3):420–441, 1989. URL: `http://dx.doi.org/10.1305/ndjfl/1093635158`, `doi:10.1305/ndjfl/1093635158`.

17 Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

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

19 G. Pottinger. A type assignment for the strongly normalizable λ-terms. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, 1980.

20 D. Prawitz. Ideas and results in proof theory. In *Proceedings of the Second Scandinavian Logic Symposium*. North-Holland, 1971.

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 J. B. Wells, A. Dimock, R. Muller, and F. A. Turbak. A calculus with polymorphic and polyvariant flow types. *J. Funct. Program.*, 12(3):183–227, 2002.

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

Appendix for referees

Type assignment system $\Lambda_{\mathsf{u}}^{\cap\cup}$

The type assignment system $\Lambda_{\mathsf{u}}^{\cap\cup}$ for a type-free (*à la* Curry) lambda-calculus featuring intersection and union types [2] is presented in Figure 1. The main properties of the system $\Lambda_{\mathsf{u}}^{\cap\cup}$ are:

▶ **Theorem 17** (Main properties of $\Lambda_{\mathsf{u}}^{\cap\cup}$ [2])**.**

Characterization. *The terms typable without use of the ω rule are precisely the strongly normalizing terms.* ◀

Parallel reduction. *If $B \vdash M : \sigma$ and $M \to_{gk} N$ then $B \vdash N : \sigma$. Here $\to_{gk}$ is the well-known "Gross-Knuth" parallel reduction [10].* ◀

Proofs of Theorems

Proof of Theorem 3. By induction over the the derivation of $\Gamma \overset{@}{\vdash} M@\Delta : \sigma$. First observe that if the derivation consists of axiom (ω) then $\Delta \equiv *$ and $\sigma = \omega$ and $\wr * \wr = \Omega \sqsubseteq M$.

If the derivation ends by

$$\frac{\Gamma \overset{@}{\vdash} M@\Delta_1 : \sigma_1 \quad \Gamma \overset{@}{\vdash} M@\Delta_2 : \sigma_2}{\Gamma \overset{@}{\vdash} M@\langle \Delta_1 , \Delta_2 \rangle : \sigma_1 \cap \sigma_2} \ (\cap I)$$

then by induction we have that both $\wr \Delta_1 \wr$ and $\wr \Delta_2 \wr$ are defined and that $\wr \Delta_1 \wr \sqsubseteq M \sqsupseteq \wr \Delta_2 \wr$, therefore $\wr \langle \Delta_1 , \Delta_2 \rangle \wr = \wr \Delta_1 \wr \sqcup \wr \Delta_2 \wr$ is defined and $\wr \Delta_1 \wr \sqcup \wr \Delta_2 \wr \sqsubseteq M$ as desired.

If the derivation ends by $(\cup E)$ we reason in the same way as in case $(\cap I)$, while all other cases are immediate by induction and the fact that $\sqsubseteq$ is a pre-congruence. ◀

Proof of Theorem 4. The only if part is Theorem 3. The if part is a straightforward induction over the derivation of $\Gamma \vdash \Delta : \sigma$. ◀

Implementation

This is a sketch of the main interface of the prototype.

```
> Help;
List of commands:
Help;                                   show this help
Load file;                              load a script file
Type typename : kind;                   define a type constant in the signature
Constant cstname : type;                define a constant in the signature
Proof proofname : type;                 start an interactive proof (work in progress)
Definition name = deltaterm [: type]    define a delta-term
Print name;                             print the definition of name
Print_all;                              print the signature
Compute name;                           print the normalized form of name
Quit;                                   quit
>
```

This is an example of a simple signature.

```
Type int   : Type;
Type float : Type;
Type True  : Type;
Type False : Type;

Constant true  : True;
Constant false : False;
Constant 0     : int;
Constant S     : int -> int;

Definition church2 =
  < \f : int -> int. \ x : int. f (f x) & \ m : (int -> int) -> (int -> int). \ s : int -> int . m (m s) >;
Definition church3 =
  < \f : int -> int. \ x : int. f (f (f x)) & \ m : (int -> int) -> (int -> int). \ s : int -> int . m (m (m s)) >;

Definition add =
  \m : (int -> int) -> (int -> int). \n : (int -> int) -> (int -> int). \f : int -> int. \x : int. m f (n f x);
Definition mul =
  \m : (int -> int) -> (int -> int). \n : (int -> int) -> (int -> int). \f : int -> int . m (n f);
Definition exp =
  \m : (int -> int) -> (int -> int). \n : ((int -> int) -> (int -> int)) -> ((int -> int) -> (int -> int)). n m;

Definition churchtrue  = \ t : (True | False). \ f : (True | False). t;
Definition churchfalse = \ t : (True | False). \ f : (True | False). f;

Definition and =
  \ x : (True | False) -> ((True | False) -> (True | False)).
   \ y : (True | False) -> ((True | False) -> (True | False)).
    \ t : (True | False).
     \ f : (True | False).
       x (y t f) f;

Definition or =
  \ x : (True | False) -> ((True | False) -> (True | False)).
   \ y : (True | False) -> ((True | False) -> (True | False)).
    \ t : (True | False).
     \ f : (True | False).
       x t (y t f);

Definition not =
  \ x : (True | False) -> ((True | False) -> (True | False)).
   \ t : (True | False).
    \ f : (True | False).
      x f t;

Definition xor =
```

```
  \ x : (True | False) -> ((True | False) -> (True | False)).
   \ y : (True | False) -> ((True | False) -> (True | False)).
     and (or x y) (not (and x y));

Definition  5   = add proj_l church2 proj_l church3 S 0;
Definition 10   = mul (add proj_l church2 proj_l church3) proj_l church2 S 0;
Definition  8   = exp proj_l church2 proj_r church3 S 0;
Definition test = xor churchtrue churchfalse inj_l true inj_r false;
```