A Decidable Subtyping Logic for Intersection and Union Types

Luigi Liquori, Claude Stolze

A Decidable Subtyping Logic for Intersection and Union Types[⋆]

Luigi Liquori and Claude Stolze

Université Côte d'Azur, INRIA, France
Luigi.Liquori@inria.fr   Claude.Stolze@inria.fr

Abstract. Using Curry-Howard isomorphism, we extend the typed lambda-calculus with intersection and union types, and its corresponding proof-functional logic, previously defined by the authors, with subtyping and explicit coercions. We show the extension of the lambda-calculus to be isomorphic to the Barbanera-Dezani-de'Liguoro type assignment system and we provide a sound interpretation of the proof-functional logic with the $\mathsf{NJ}(\beta)$ logic, using Mints' realizers.
We finally present a sound and complete algorithm for subtyping in presence of intersection and union types. The algorithm is conceived to work for the (sub)type theory Ξ.

Keywords. Logics and lambda-calculus, type and subtype systems.

1 Introduction

This paper is a contribution to the study of typed lambda-calculi *à la* Church in presence of intersection, union types, and subtyping and their role in logical investigations; it is a natural follow up of a recent paper by the authors [DdLS16].

Intersection types were first introduced as a form of *ad hoc* polymorphism in (pure) lambda-calculi *à la* Curry. The paper by Barendregt, Coppo, and Dezani [BCDC83] is a classic reference, while [Bar13] is a definitive reference.

Union types were later introduced as a dual of intersection by MacQueen, Plotkin, and Sethi [MPS86]: Barbanera, Dezani, and de'Liguoro [BDCd95] is a definitive reference; Frisch, Castagna, and Benzaken [FCB08] designed a type system with intersection, union, and negation types whose semantics are loosely the corresponding set-theoretical constructs.

As intersection and union types had their classical development for (undecidable) type assignment systems, many papers moved from intersection and union type theories to (typed) lambda-calculi *à la* Church: the programming language Forsythe, by Reynolds [Rey96], is probably the first reference for intersection types, while Pierce's PhD thesis combines also unions and intersections [Pie91]; a recent implementation of a typed programming language featuring intersection and union types is [Dun14].

Proof-functional logical connectives allow reasoning about the structure of logical proofs, in this way giving to the latter the status of first-class objects. This is in contrast with classical truth-functional connectives where the meaning of a compound formula

⋆ Work supported by the COST Action CA15123 EUTYPES "The European research network on types for programming and verification".

is dependent only on the truth value of its subformulas. Following this approach, the logical relation between type assignment systems and typed systems featuring intersection and union types were studied in [LR07,DL10,DdLS16].

Proof-functional connectives represent evidence as a "polymorphic" construction, that is, the same evidence can be used as a proof for different sentences. Pottinger [Pot80] first introduced a conjunction, called strong conjunction $\cap$, requiring more than the existence of constructions proving the left and the right hand side of the conjuncts. According to Pottinger: *"The intuitive meaning of $\cap$ can be explained by saying that to assert $A \cap B$ is to assert that one has a reason for asserting A which is also a reason for asserting B"*. This interpretation makes inhabitants of $A \cap B$ as uniform evidences for both A and B. Later, Lopez-Escobar [LE85] presented the first proof-functional logic with strong conjunction as a special case of ordinary conjunction.

Mints [Min89] presented a logical interpretation of strong conjunction using *realizers*: the logical predicate $r_{A\cap B}[M]$ is true if the pure lambda-term M is a realizer (also read as "M is a method to assess $A \cap B$") for both the formula $r_A[M]$ and $r_B[M]$.

Inspired by this, Barbanera and Martini tried to answer the question of realizing other "proof-functional" connectives, like strong implication, Lopez-Escobar's strong equivalence, or Bruce, Di Cosmo, and Longo provable type isomorphism [BCL92].

Recently [DdLS16] extended the logical interpretation with union types as another proof-functional operator, the strong union $\cup$. Paraphrasing Pottinger's point of view, we could say that the intuitive meaning of $\cup$ is that if we have a reason to assert A (or B), then the same reason will also assert $A \cup B$. This interpretation makes inhabitants of $(A \cup B) \supset C$ be uniform evidences for both $A \supset C$ and $B \supset C$. Symmetrically to intersection, and extending Mints' logical interpretation, the logical predicate $r_{A\cup B}[M]$ succeeds if the pure lambda-term M is a realizer for either the formula $r_A[M]$ or $r_B[M]$.

1.1 Contributions.

This paper focus on the logical and algorithmic aspects of subtyping in presence of intersection and union types: our interest is not only theoretical but also pragmatic since, in a dependent-type setting, it opens the door to logical frameworks and proof-assistants. We also inspect the relationship between pure and typed lambda-calculi and their corresponding proof-functional logics as dictated by the well-known Curry-Howard [How80] correspondence. We'll present and explore the relationships between the following four formal systems:

- $\Lambda^{\cap\cup}_{\mathsf{u}\leqslant}$, the type assignment system with intersection and union types for pure lambda-calculus with subtyping with the (sub)type theory Ξ, as defined in [BDCd95]: a type assignment judgment have the shape $\Gamma \vdash M : \sigma$;
- $\Lambda^{\cap\cup}_{\mathsf{t}\leqslant}$, an extension of the typed lambda-calculus with strong pairs and strong sums $\Lambda^{\cap\cup}_{\mathsf{t}}$, as defined in [DL10], with subtyping and explicit coercions: a type judgment has the shape $\Gamma^{@} \vdash M@\Delta : \sigma$, where Δ is a typed lambda-term enriched with strong pairs and strong sums;
- $\mathcal{L}^{\cap\cup}_{\leqslant}$, an extension of the proof-functional logic $\mathcal{L}^{\cap\cup}$ of [DdLS16] with *ad hoc* formulas and inference rules for subtyping and explicit coercions: sequents have the shape $\Gamma \vdash \Delta : \sigma$;

– $\mathsf{NJ}(\beta)$, a natural deduction system for derivations in first-order intuitionistic logic with pure lambda-terms [Pra65].

Intuitively, Δ denotes a proof for a type assignment derivation for M; from an operational point of view, reductions in pure M and typed Δ must be synchronized by suitable parallel reduction rules in order to preserve parallel reduction of subjects. From a typing point of view, the type rules of $\Lambda^{\cap\cup}_{\mathsf{t}\leqslant}$ should encode the proof-functional nature of strong intersection and strong union, *i.e.* the fact that in an intersection (resp. union) the two Δ relate to the same M.

Thanks to an erasing function $\wr - \wr$ translating typed Δ to pure M, we could reason only on a proof-functional logic $\mathcal{L}^{\cap\cup}_{\leqslant}$ assigning types to Δ. Therefore, the original contribution are as follows:

– to define the typed lambda-calculus $\Lambda^{\cap\cup}_{\mathsf{t}\leqslant}$ obtained by extending the typed calculus of [DL10] with a subtyping relation and explicit coercions, keeping decidability of type checking, and showing the isomorphism with the type assignment system $\Lambda^{\cap\cup}_{\mathsf{u}\leqslant}$ of [BDCd95]. Terms of $\Lambda^{\cap\cup}_{\mathsf{t}\leqslant}$ have the form $M@\Delta$ where M is a pure lambda-term, while Δ is a typed lambda-term enriched with strong pairs and and strong sums;
– to define $\mathcal{L}^{\cap\cup}_{\leqslant}$ obtained by extending the proof-functional logic $\mathcal{L}^{\cap\cup}$ of [DdLS16]: we show that the extended $\mathcal{L}^{\cap\cup}_{\leqslant}$ logic of subtyping is sound with respect to the realizability logic $\mathsf{NJ}(\beta)$, using Mint's realizability arguments;
– to present an algorithm for subtyping in presence of intersection and union types. The algorithm (presented in functional style) is conceived to work for the (sub)type theory Ξ (*i.e.* axioms 1 to 14, as presented in [BDCd95]).

For lack of space, the full metatheoretical development can be found in [LS17].

1.2 Related Work

We shortly list the main research lines involving type (assignment) systems with intersection, union, and subtyping for (un)typed lambda-calculi, proof-functional logics containing “strong operators”, and realizability.

The formal investigation of soundness and completeness for a notion of realizability was initiated by Lopez-Escobar [LE85] and subsequently refined by Mints [Min89].

Barbanera and Martini [BM94] studied three proof-functional operators, namely the strong conjunction, the relevant implication (related with Meyer-Routley's [MR72] system B^+), and the strong equivalence connective for double implication, relating those connectives with a suitable type assignment system, a realizability semantics, and a completeness theorem.

Dezani-Ciancaglini, Ghilezan, and Venneri [DCGV97] investigated a Curry-Howard interpretation of intersection and union types (for Combinatory Logic): using the well-understood relation between combinatory logic and lambda-calculus, they encode type-free lambda-terms into suitable combinatory logic formulas and then type them using intersection and union types. This is a complementary approach to the realizability-based one here and in [DdLS16].

Various authors defined lambda-calculi *à la* Church for intersection types with related logics: see [Bar13] (pp. 780-781) for a complete list.

As mentioned before, Barbanera, Dezani-Ciancaglini, and de'Liguoro [BDCd95] introduced a pure lambda-calculus $\Lambda^{\cap\cup}_{\mathsf{u}\leqslant}$ with a related type assignment system featuring intersection and union types, and a powerful subtyping relation.

The previous work [DL10] presented a typed calculus $\Lambda^{\cap\cup}_{\mathsf{t}}$ (without subtyping) that explored the relationship between the proof-functional intersections and unions and the corresponding type assignment system (without subtyping).

In [DdLS16] we introduced an erasing function, called *essence* and denoted by $\wr\varDelta\wr$, to understand the connection between pure terms and typed terms: we proved the isomorphism between $\Lambda^{\cap\cup}_{\mathsf{t}}$ and $\Lambda^{\cap\cup}_{\mathsf{u}}$, and we showed that $\mathcal{L}^{\cap\cup}$ can be thought of as a proof-functional logic. The present paper extends all the systems and logics of [DdLS16] and presents a comparative analysis of the (sub)type theories $\varXi$ and $\varPi$ of [BDCd95]: this motivates the use of the (sub)type theory $\varXi$ with their natural correspondence with $\mathsf{NJ}(\beta)$.

Hindley gave first a subtyping algorithm for type intersection [Hin82]: there is a rich literature reducing the subtyping problem in presence of intersection and union to a set constraint problem: good references are [Dam94,Aik99,DP04,FCB08]. The closest work to the algorithm presented in this paper has been made by Aiken and Wimmers [AW93] who designed an algorithm whose input is a list of set constraints with unification variables, usual arrow types, intersection, complementation, and constructor types. Their algorithm first rewrites types in disjunctive normal form, then simplifies the constraints until it shows the system has no solution, or until it can safely unify the variables. The rewriting in disjunctive normal form makes this algorithm exponential in time and space in the worst case.

Pfenning work on Refinement Types [Pfe93] pioneered an extension of Edinburgh Logical Framework with subtyping and intersection types: our aim is to study extensions of LF featuring fully fledged proof-functional logical connectives like strong conjunction, strong disjunction in presence of subtyping and relevant implication.

2 System

The pseudo-syntax of σ, M, $\varDelta$, and the derived $M@\varDelta$ are defined using the following three syntactic categories:

$$\begin{aligned}
\sigma &::= \omega \mid \phi \mid \sigma \to \sigma \mid \sigma \cap \sigma \mid \sigma \cup \sigma\\
M &::= x \mid \lambda x.M \mid M\,M\\
\varDelta &::= * \mid x \mid \lambda x{:}\sigma.\varDelta \mid \varDelta\,\varDelta \mid \langle \varDelta\,,\varDelta\rangle \mid [\varDelta\,,\varDelta] \mid \mathsf{pr}_1\,\varDelta \mid \mathsf{pr}_2\,\varDelta \mid \mathsf{in}_1\,\varDelta \mid \mathsf{in}_2\,\varDelta \mid [\sigma]\varDelta
\end{aligned}$$

where ϕ denotes arbitrary constant types and ω denotes a special type that is inhabited by all terms. The $\varDelta$-expression $\langle\varDelta\,,\varDelta\rangle$ denotes the strong pair while $[\varDelta\,,\varDelta]$ denotes the strong sum, with the respective projections and injections, respectively. Finally $[\sigma]\varDelta$ denotes the explicit coercion of $\varDelta$ with the type σ.

The untyped reduction semantics for the calculus *à la* Curry $\Lambda^{\cap\cup}_{\mathsf{t}}$ is ordinary β-reduction, even if subject reduction holds only in presence of the "Gross-Knuth" parallel reduction (Def. 13.2.7 in [Bar84]), where all redexes in M are contracted simultaneously. Reduction for the calculus *à la* Church $\Lambda^{\cap\cup}_{\mathsf{t}}$ is delicate because it must keep

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

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

Fig. 1. Intersection and Union Type Assignment System $\Lambda_{\mathsf{u}}^{\cap\cup}$ [BDCd95] (main rules).

$$\frac{\Gamma, x{:}\sigma_1 \vdash M@\Delta : \sigma_2}{\Gamma \vdash \lambda x.M@\lambda x{:}\sigma_1.\Delta : \sigma_1 \to \sigma_2}\ (\to I) \qquad \frac{\Gamma \vdash M@\Delta : \sigma_i \quad i \in \{1,2\}}{\Gamma \vdash M@\mathsf{in_i}\,\Delta : \sigma_1 \cup \sigma_2}\ (\cup I_i)$$

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

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

Fig. 2. Typed Calculus $\Lambda_{\mathsf{t}}^{\cap\cup}$ [DL10] (main rules).

synchronized the untyped reduction of M with the typed reduction of Δ: it is defined in Section 5 of [DL10]. Reductions in $\mathcal{L}^{\cap\cup}$ is ordinary β-reduction plus the following four reduction rules:

$$\mathsf{pr_i}\,\langle \Delta_1 \,, \Delta_2 \rangle \longrightarrow_{\mathsf{pr_i}} \Delta_i \qquad [\lambda x{:}\sigma_1.\Delta_1 \,, \lambda x{:}\sigma_2.\Delta_2]\,\mathsf{in_i}\,\Delta_3 \longrightarrow_{\mathsf{in_i}} \Delta_i\{\Delta_3/\iota\} \quad i \in \{1,2\}$$

Figure 1 presents the main rules of the type assignment system of [BDCd95]: note that the type inference rules are not syntax-directed. Figure 2 presents the main rules of the typed calculus $\Lambda_{\mathsf{t}}^{\cap\cup}$ of [DL10][1]; note that this type system is completely syntax directed.

The next definition clarifies what we intend with "correspondence" between an untyped M and a typed Δ: the essence partial function shows the syntactic relation between type free and typed lambda-terms. Essence maps typed proof-terms (Δ's) into pure λ-terms: intuitively, two typed Δ-terms prove the same formula if they have the same proof-essence.

Definition 1 (Proof Essence).
The essence function between pure and typed lambda-terms is defined as follows:

$$\begin{array}{rlrl}
\wr x \wr & \triangleq x & \wr \lambda x{:}\sigma.\Delta \wr & \triangleq \lambda x.\wr \Delta \wr \\
\wr \Delta_1\,\Delta_2 \wr & \triangleq \wr \Delta_1 \wr \wr \Delta_2 \wr & \wr [\sigma]\Delta \wr & \triangleq \wr \Delta \wr \\
\wr \mathsf{pr_i}\,\Delta \wr & \triangleq \wr \Delta \wr & \wr \mathsf{in_i}\,\Delta \wr & \triangleq \wr \Delta \wr \\
\wr \langle \Delta_1 \,, \Delta_2 \rangle \wr & \triangleq \wr \Delta_1 \wr & \textit{if}\ \wr \Delta_1 \wr & \equiv \wr \Delta_2 \wr \\
\wr [\lambda x{:}\sigma_1.\Delta_1 \,, \lambda x{:}\sigma_2.\Delta_2)]\,\Delta_3 \wr & \triangleq \wr \Delta_1 \wr \{\wr \Delta_3 \wr / x\} & \textit{if}\ \wr \Delta_1 \wr & \equiv \wr \Delta_2 \wr
\end{array}$$

[1] Contexts $\Gamma^{@}$ contains assumptions of the shape $x@\iota_x{:}\sigma$: the present paper uses ordinary contexts Γ, since Γ can be easily obtained by erasing all the $\_@\iota_x$ in $\Gamma^{@}$.

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

$$\frac{\begin{array}{l}\Gamma \vdash \Delta_1 : \sigma_1 \\ \Gamma \vdash \Delta_2 : \sigma_2 \quad \wr\Delta_1\wr \equiv \wr\Delta_2\wr\end{array}}{\Gamma \vdash \langle \Delta_1 \,, \Delta_2 \rangle : \sigma_1 \cap \sigma_2}\ (\cap I) \qquad \frac{\Gamma \vdash \Delta : \sigma_1 \cap \sigma_2 \quad i \in \{1,2\}}{\Gamma \vdash \mathsf{pr_i}\, \Delta : \sigma_i}\ (\cap E_i)$$

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

Fig. 3. Proof-functional logic $\mathcal{L}^{\cap\cup}$ (main rules).

Figure 3 presents the main rules of the proof-functional logic $\mathcal{L}^{\cap\cup}$ of [DdLS16]: that logic is *proof-functional*, in the sense of Pottinger [Pot80] and Lopez-Escobar [LE85]: formulas encode, using the Curry-Howard isomorphism, *derivations* $\mathcal{D} : \Gamma \vdash M : \sigma$ in the type assignment system $\Lambda_{\mathsf{u}}^{\cap\cup}$ which are, in turn, isomorphic to typed *judgments* $\Gamma \vdash M@\Delta : \sigma$ of $\Lambda_{\mathsf{t}}^{\cap\cup}$. It is worth noticing that if we drop the restriction concerning the "essence" in rules $(\cap I)$ and $(\cup E)$ in the system $\mathcal{L}^{\cap\cup}$ and replace $\sigma \cap \tau$ by $\sigma \times \tau$, and $\sigma \cup \tau$ by $\sigma + \tau$, we get a simply typed lambda-calculus with product and sums, namely a truth-functional intuitionistic propositional logic with implication, conjunction, and disjunction in disguise: the resulting logic loses its proof-functionality.

The whole picture is now ready to be extended with the subtyping relation, as introduced in [BCDC83] and extended in [BDCd95] with unions. Subtyping is a preorder over types, and it is written as $\sigma \leqslant \tau$; a (sub)type theory denotes any collection of inequalities between types satisfying natural closure conditions. The (sub)type theory, called Ξ (see Definition 3.6 of [BDCd95]), is defined by the subtyping axioms and inference rules defined as follows:

(1)	$\sigma \leqslant \sigma \cap \sigma$	(8)	$\sigma_1 \leqslant \sigma_2, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \cup \tau_1 \leqslant \sigma_2 \cup \tau_2$
(2)	$\sigma \cup \sigma \leqslant \sigma$	(9)	$\sigma \leqslant \tau, \tau \leqslant \rho \Rightarrow \sigma \leqslant \rho$
(3)	$\sigma \cap \tau \leqslant \sigma, \sigma \cap \tau \leqslant \tau$	(10)	$\sigma \cap (\tau \cup \rho) \leqslant (\sigma \cap \tau) \cup (\sigma \cap \rho)$
(4)	$\sigma \cup \tau \leqslant \sigma, \sigma \cup \tau \leqslant \tau$	(11)	$(\sigma \to \tau) \cap (\sigma \to \rho) \leqslant \sigma \to (\tau \cap \rho)$
(5)	$\sigma \leqslant \omega$	(12)	$(\sigma \to \rho) \cap (\tau \to \rho) \leqslant (\sigma \cup \tau) \to \rho$
(6)	$\sigma \leqslant \sigma$	(13)	$\omega \leqslant \omega \to \omega$
(7)	$\sigma_1 \leqslant \sigma_2, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \cap \tau_1 \leqslant \sigma_2 \cap \tau_2$	(14)	$\sigma_2 \leqslant \sigma_1, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \to \tau_1 \leqslant \sigma_2 \to \tau_2$

The (sub)theory Ξ suggests the interpretation of ω as the *set universe*, of $\cap$ as the *set intersection*, of $\cup$ as the *set union*, and of $\leqslant$ as a sound (but not complete) *subset relation*, respectively, in the spirit of [FCB08]. In the following, we write $\sigma \sim \tau$ iff $\sigma \leqslant \tau$ and $\tau \leqslant \sigma$. We note that distributivity of union over intersection and intersection over union, *i.e.* $\sigma \cup (\tau \cap \rho) \sim (\sigma \cup \tau) \cap (\sigma \cup \rho)$ and $\sigma \cap (\tau \cup \rho) \sim (\sigma \cap \tau) \cup (\sigma \cap \rho)$ are derivable (see, *e.g.* derivation in [BDCd95], page 9).

Once the subtyping preorder has been defined, a classical subsumption (respectively an explicit coercion rule) can be defined as follows:

$$\frac{G \vdash_{\mathsf{NJ}} A \quad G \vdash_{\mathsf{NJ}} B}{G \vdash_{\mathsf{NJ}} A \wedge B}\ (\wedge I) \qquad \frac{G \vdash_{\mathsf{NJ}} A_1 \wedge A_2 \quad i=1,2}{G \vdash_{\mathsf{NJ}} A_i}\ (\wedge E_i)$$

$$\frac{G \vdash_{\mathsf{NJ}} A_i \quad i=1,2}{G \vdash_{\mathsf{NJ}} A_1 \vee A_2}\ (\vee I_i) \qquad \frac{\begin{array}{l} G, A \vdash_{\mathsf{NJ}} C \\ G, B \vdash_{\mathsf{NJ}} C \quad G \vdash_{\mathsf{NJ}} A \vee B \end{array}}{G \vdash_{\mathsf{NJ}} C}\ (\vee E)$$

Fig. 4. The Logic NJ (main rules)

$$\frac{\Gamma \vdash M : \sigma \quad \sigma \leqslant \tau}{\Gamma \vdash M : \tau}\ (\leqslant) \qquad \frac{\Gamma \vdash M@\Delta : \sigma \quad \sigma \leqslant \tau}{\Gamma \vdash M@[\tau]\Delta : \tau}\ (\leqslant) \qquad \frac{\Gamma \vdash \Delta : \sigma \quad \sigma \leqslant \tau}{\Gamma \vdash [\tau]\Delta : \tau}\ (\leqslant)$$

This completes the reminder of the type assignment $\Lambda_{\mathsf{u}\leqslant}^{\cap\cup}$ of [BDCd95], and the presentation of the typed system $\Lambda_{\mathsf{t}\leqslant}^{\cap\cup}$, and of the proof-functional logic $\mathcal{L}_{\leqslant}^{\cap\cup}$, respectively.

The next theorem relates the three systems: the key concept is the essence partial map $\wr - \wr$ that allows to interpret union, intersection, and explicit coercions as proof-functional connectives.

Theorem 2 (Equivalence).
Let M and Δ and $\overset{@}{\Gamma}, \Gamma, B$ such that $\wr\Delta\wr \equiv M$. Then:

1. *$\Gamma \vdash M : \sigma$ iff $\overset{@}{\Gamma} \vdash M@\Delta : \sigma$;*
2. *$\overset{@}{\Gamma} \vdash M@\Delta : \sigma$ iff $\Gamma \vdash \Delta : \sigma$;*
3. *$\Gamma \vdash M : \sigma$ iff $\Gamma \vdash \Delta : \sigma$.*

Proof. Point 1 by upgrading Theorem 10 of [DL10]; point 2 by induction on the structure of derivations, using Definition 1; point 3 by 1,2. □

The next theorem states that adding subtyping as explicit coercions does not break the properties of the extended typed systems.

Theorem 3 (Conservativity).
The typed system $\Lambda_{\mathsf{t}\leqslant}^{\cap\cup}$ and the proof-functional logic $\mathcal{L}_{\leqslant}^{\cap\cup}$, both obtained by extending with the (sub)type theory Ξ and with explicit coercions type rules $(\leqslant)$, preserve subject reduction (parallel-synchronized β-reduction for $\Lambda_{\mathsf{t}\leqslant}^{\cap\cup}$), Church-Rosser, strong normalization, unicity of typing, decidability of type reconstruction and of type checking, judgment decidability and isomorphism of typed-untyped derivations.

Proof. For proving properties of $\Lambda_{\mathsf{t}\leqslant}^{\cap\cup}$ we proceeds by upgrading results of Theorems 11, 12 and 19 of [DL10] with the subsumption rule $(\leqslant)$. Properties of $\mathcal{L}_{\leqslant}^{\cap\cup}$ are mostly inherited by $\Lambda_{\mathsf{t}\leqslant}^{\cap\cup}$ using Theorem 2 or, as for case of subject reduction for β-, $\mathsf{pr_i}$- and $\mathsf{in_i}$-reductions, is proved by induction on the structure of the derivation. Decidability of subtyping is proved in Theorems 18 and 19. □

3 Realizers

We start this section by recalling the logic $\vdash_{\mathsf{NJ}}$, as sketched in Figure 4. By NJ we mean the natural deduction presentation of the intuitionistic first-order predicate calculus [Pra65]. Derivations in NJ are trees of judgments $G \vdash_{\mathsf{NJ}} A$, where G is a set of undischarged assumptions, rather than trees of formulas, as in Gentzen's original formulation. Then we extend NJ as follows:

Definition 4. (Logic $\mathsf{NJ}(\beta)$).
Let $\mathbf{P}_\phi(x)$ be a unary predicate for each atomic type ϕ: the natural deduction system for first-order intuitionistic logic $\mathsf{NJ}(\beta)$ extends NJ with untyped lambda-terms and predicates $\mathbf{P}_\phi(x)$, the latter being axiomatized via the two Post rules:

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

For a given context $\Gamma \triangleq \{x_1{:}\sigma_1, \dots, x_n{:}\sigma_n\}$, we associate a logical context $G_\Gamma \triangleq r_{\sigma_1}[x_1], \dots, r_{\sigma_n}[x_n]$. Note that $G_{\Gamma,x:\sigma} \equiv G_\Gamma, r_\sigma[x]$ and $x \notin \mathsf{Fv}(G_\Gamma)$, since $x \notin \mathsf{Dom}(\Gamma)$, by context definition.

In [DdLS16], we provided a foundation for the proof-functional logic $\mathcal{L}^{\cap\cup}$ by extending Mints' provable realizability to cope with intersection and union types, but without subtyping. What follows scale up Mints' realizability to $\mathcal{L}^{\cap\cup}_{\leqslant}$. The next definition is a reminder of the notion of realizer, as first introduced for intersection types by Mints [Min89], and extended by the authors in [DdLS16].

Definition 5. (Mints' realizers in $\mathsf{NJ}(\beta)$).
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 follows:

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

where $\supset$ denotes implication, $\wedge$ and $\vee$ are the logical connectives for conjunction and disjunction respectively, that must be kept distinct from $\cap$ and $\cup$. Formulas have the shape $r_\sigma[M]$, whose intended meaning is that M is a method for σ in the intersection-union type discipline with subtyping.

Intuitively, we write $r_\sigma[M]$ to denote a formula in $\mathsf{NJ}(\beta)$, realized by the pure lambda-term M of type σ in $\Lambda^{\cap\cup}_{\mathsf{u}\leqslant}$. Observe that M is "distilled" by applying the essence function to the typed proof-term Δ, which faithfully encodes the type assignment derivation $\Gamma \vdash \wr\Delta\wr : \sigma$ in $\Lambda^{\cap\cup}_{\mathsf{u}\leqslant}$. The next theorem states that the proof-functional logic $\mathcal{L}^{\cap\cup}_{\leqslant}$ is sound *w.r.t.* Mints' realizers in $\mathsf{NJ}(\beta)$.

Lemma 6 **($\Lambda^{\cap\cup}_{\mathsf{u}\leqslant}$ *versus* $\mathsf{NJ}(\beta)$).** *If $\Gamma \vdash M : \sigma$, then $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$.*

Proof. By structural induction on the derivation tree of $B \vdash M : \sigma$:
- *rules (Var), $(\cup I)$, $(\cap I)$, $(\cap E)$ correspond trivially to (Hyp), $(\vee I)$, $(\wedge I)$, and $(\wedge E)$;*
- *rule $(\cup E)$ is derivable from rule $(\vee E)$ and a classical substitution lemma;*
- *it can be showed that all the subtyping rules are derivable in $\mathsf{NJ}(\beta)$, therefore $(\leqslant)$ is derivable;*

– *rules* $(\to I)$ *and* $(\to E)$ *are derivable:*

$$\dfrac{\dfrac{G_\Gamma, r_\sigma[x] \vdash_{\mathsf{NJ}(\beta)} r_\tau[M]}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[x] \supset r_\tau[M]}\,(\supset I)}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma\to\tau}[\lambda x.M]}\,(\forall I)$$

$$\dfrac{\dfrac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma\to\tau}[M]}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[N] \supset r_\tau[MN]}\,(\forall E) \quad G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[N]}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\tau[MN]}\,(\supset E)$$

□

Informally speaking, $r_\sigma[M]$ can be interpreted as "M is an element of the set σ", and the judgment $\sigma_1 \leqslant \sigma_2$ in the (sub)type theory Ξ can be interpreted as $r_{\sigma_1}[x] \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_2}[x]$. As a simple consequence of Lemma 6, we can now state soundness:

Theorem 7 (Soundness of $\mathsf{NJ}(\beta)$ and $\mathcal{L}^{\cap\cup}_{\leqslant}$). *If $\Gamma \vdash \Delta : \sigma$ then $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[\wr\Delta\wr]$.*

Proof. *Trivial by Lemma 6 and Theorem 2 part 3.* □

The completeness result, *i.e.* If $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$, then there exists Δ such that $\Gamma \vdash \Delta : \sigma$ and $\wr\Delta\wr \equiv M$ is more tricky because of the presence of the union elimination rule $(\vee E)$ in $\mathsf{NJ}(\beta)$. As an example, let $\phi \equiv (\sigma \cup \tau) \cap (\sigma \cup \rho) \to \sigma \cup (\tau \cap \rho)$: with a fairly complex derivation in $\mathsf{NJ}(\beta)$ we can realize $G_\emptyset \vdash_{\mathsf{NJ}(\beta)} r_\phi[\lambda x.x]$, and then by completeness the type assignment $\emptyset \vdash \lambda x.x : \phi$ should be derivable in [BDCd95], which is not the case without subtyping. We left completeness for a future work.

Remark 8.
The type assignment system $\Lambda^{\cap\cup}_{\mathsf{u}\leqslant}$ of [BDCd95] was based on the (sub)type theory Ξ (see Definition 3.6 of [BDCd95]): the paper also introduced a stronger (sub)type theory, called Π, by adding the extra axiom

$$(15) \qquad \mathbf{P}(\sigma) \Rightarrow \sigma \to \tau \cup \rho \leqslant (\sigma \to \tau) \cup (\sigma \to \rho),$$

where $\mathbf{P}(\sigma)$ is true if σ syntactically corresponds to an Harrop formula. However, in $\mathsf{NJ}(\beta)$, the judgment $r_{\sigma\to(\tau\cup\rho)}[x] \vdash_{\mathsf{NJ}(\beta)} r_{(\sigma\to\tau)\cup(\sigma\to\rho)}[x]$ is not derivable because the judgment $A \supset (B \vee C) \vdash_{\mathsf{NJ}(\beta)} (A \supset B) \vee (A \supset C)$ is not derivable in NJ. As such, the (sub)type theory Π cannot be overlapped with an interpretation of (sub)types as (sub)sets, as the following example show. The identity function $\lambda x.x$ inhabits the function set $\{a, b\} \to \{a\} \cup \{b\}$ but, by axiom (15), it should also inhabit $\{a, b\} \to \{a\}$ or $\{a, b\} \to \{b\}$, which is clearly not the case.

4 Subtyping algorithm

The previous section showed that the proof-functional logic $\mathcal{L}^{\cap\cup}_{\leqslant}$ is sound *w.r.t.* the logic $\mathsf{NJ}(\beta)$. The truth of the sequent "$\Gamma \vdash \Delta : \sigma$" complicates its decidability because of the presence of the predicate $\sigma \leqslant \tau$ as a premise in rule $(\leqslant)$: in fact, the subtype system is not an algorithm because of the presence of reflexivity and transitivity rules that are not syntax-directed. The same subtyping premise can affect the decidability of type checking of $\Lambda^{\cap\cup}_{\mathtt{t}\leqslant}$. This section presents a sound and complete algorithm $\mathcal{A}$ for subtyping in the (sub)type theory Ξ. In what follows we use the following useful shorthands:

$$\cap_i(\cup_j\sigma_{i,j}) \triangleq \cap_1(\cup_1\sigma_{1,1}\ldots\cup_j\sigma_{1,j})\ldots\cap_i(\cup_1\sigma_{i,1}\ldots\cup_j\sigma_{i,j}), \text{and}$$

$$\cup_i(\cap_j\sigma_{i,j}) \triangleq \cup_1(\cap_1\sigma_{1,1}\ldots\cap_j\sigma_{1,j})\ldots\cup_i(\cap_1\sigma_{i,1}\ldots\cap_j\sigma_{i,j}).$$

Those shorthands can also apply to unions of unions, intersections of intersections, intersections of arrows, etc.

Algorithm $\mathcal{A}$ alone has a polynomial complexity, but it requires the types to be in some normal form that will be detailed later. We therefore have a preprocessing phase that is exponential in space. The preprocessing uses the following four subroutines:

– $\mathcal{R}_1$, to simplify the shape of types containing the ω type: its complexity is linear;
– $\mathcal{R}_2$ (well-known), to transform a type in its conjunctive normal form, denoted by CNF, *i.e.* types being, roughly, intersection of unions: its complexity is exponential in space;
– $\mathcal{R}_3$ (well-known), to transform a type in its disjunctive normal form, denoted by DNF, *i.e.* types being, roughly, union of intersections: its complexity is exponential in space;
– $\mathcal{R}_4$, to transform a type in its arrow normal form, denoted by ANF, *i.e.* types being, roughly, arrow types where all the domains are intersection of ANF and all the codomains are union of ANF: its complexity is exponential in space.

Definition 9. ***(Subroutine*** $\mathcal{R}_1$***)***
The term rewriting system $\mathcal{R}_1$ *is defined as follows:*
- $\omega\cap\sigma$ *and* $\sigma\cap\omega$ *rewrite to* σ*;*
- $\omega\cup\sigma$ *and* $\sigma\cup\omega$ *rewrite to* ω*;*
- $\sigma\to\omega$ *rewrites to* ω*.*

It is easy to verify that $\mathcal{R}_1$ terminates and his complexity is linear. The next definition recall the usual conjunctive/disjunctive normal form with corresponding subroutines $\mathcal{R}_2$ and $\mathcal{R}_3$, and introduce the arrow normal form with his corresponding subroutine $\mathcal{R}_4$.

Definition 10. (Subroutines $\mathcal{R}_2$ and $\mathcal{R}_3$)
– *A type is in CNF if it has the form* $\cap_i(\cup_j\sigma_{i,j})$*, and all the* $\sigma_{i,j}$ *are either atomic types, arrow types, or* ω*;*
– *The term rewriting system* $\mathcal{R}_2$ *rewrites a type in its CNF; it is defined as follows:*
  - $\sigma\cup(\tau\cap\rho)$ *rewrites to* $(\sigma\cup\tau)\cap(\sigma\cup\rho)$*;*
  - $(\sigma\cap\tau)\cup\rho$ *rewrites to* $(\sigma\cup\rho)\cap(\tau\cup\rho)$*;*
– *A type is in DNF if it has the form* $\cup_i(\cap_j\sigma_{i,j})$*, and all the* $\sigma_{i,j}$ *are either atomic types, arrow types, or* ω*;*
– *The term rewriting system* $\mathcal{R}_3$ *rewrites a type in its DNF; it is defined as follows:*
  - $\sigma\cap(\tau\cup\rho)$ *rewrites to* $(\sigma\cap\tau)\cup(\sigma\cap\rho)$*;*
  - $(\sigma\cup\tau)\cap\rho$ *rewrites to* $(\sigma\cap\rho)\cup(\tau\cap\rho)$*.*

It is well documented in the literature that $\mathcal{R}_2$ and $\mathcal{R}_3$ terminate, and that the complexity of those algorithms is exponential.

As you can see in the (sub)type Ξ's rules (11) and (12), intersection and union interact with the arrow type; in order to simplify this, we define the following subroutine:

Definition 11. (Subroutine $\mathcal{R}_4$)
- *A type is in* arrow normal form *(ANF) if :*
  - *it is an atomic type or ω;*
  - *it is an arrow type in the form $(\cap_i \sigma_i) \to (\cup_j \tau_j)$, where the σ_i and τ_j are ANFs;*
- *The term rewriting system $\mathcal{R}_4$ rewrites an arrow type into an* intersection *of ANF; it is defined as follows:*
  - *$\sigma \to \tau$ rewrites to $\mathcal{R}_3(\sigma) \to \mathcal{R}_2(\tau)$;*
  - *$\cup_i \sigma_i \to \cap_j \tau_j$ rewrites to $\cap_i(\cap_j(\sigma_i \to \tau_j))$.*

Since $\mathcal{R}_2$ and $\mathcal{R}_3$ terminate, $\mathcal{R}_4$ terminates and its complexity is exponential. The next lemma ensures we can safely use the $\mathcal{R}_{1,2,3,4}$ subroutines in the preprocessing, because they preserve type equivalence, denoted by $\sim$. Let $\sigma \sim \tau$ iff $\sigma \leqslant \tau$ and $\tau \leqslant \sigma$.

Lemma 12. *For all the term rewriting systems $\mathcal{R}_{1,2,3,4}$ we have that $\mathcal{R}(\sigma) \sim \sigma$.*

Proof. *Each rewriting rule rewrites a term into an equivalent ($\sim$) term.* □

We can now define how the types are being preprocessed before being fed to the algorithm $\mathcal{A}$.

Definition 13.
- *A type is in disjunctive arrow normal form (DANF) if it is in DNF and all the arrow type subterms are in ANF;*
- *A type is in conjunctive arrow normal form (CANF) if it is in CNF and all the arrow type subterms are in ANF.*

Let $\sigma \leqslant \tau$ be an instance of the subtyping problem. The preprocessing algorithm rewrites σ into a DANF by applying $\mathcal{R}_3 \circ \mathcal{R}_4 \circ \mathcal{R}_1$, and τ into a CANF by applying $\mathcal{R}_2 \circ \mathcal{R}_4 \circ \mathcal{R}_1$.

4.1 The algorithm $\mathcal{A}$

Our algorithm $\mathcal{A}$ is composed of two mutually inductive functions, called $\mathcal{A}_1$ and $\mathcal{A}_2$. It proceeds as follows: $\sigma \leqslant \tau$ is preprocessed into $\cup_i(\cap_j \sigma_{i,j}) \leqslant \cap_h(\cup_k \tau_{h,k})$, where all the $\sigma_{i,j}, \tau_{h,k}$ are in ANF; it is then processed by $\mathcal{A}_1$, which accepts or rejects it.

Definition 14. (Main function $\mathcal{A}_1$).
input: *$\cup_i(\cap_j \sigma_{i,j}) \leqslant \cap_h(\cup_k \tau_{h,k})$ where all the $\sigma_{i,j}, \tau_{h,k}$ are ANF;* ***output:*** *boolean.*
- *if $\cap_h(\cup_k \tau_{h,k})$ is ω, then accept, else*
  if for all i and h, there exists some j and some k, such that $\mathcal{A}_2(\sigma_{i,j} \leqslant \tau_{h,k})$ is true, then accept, else reject.

Definition 15. (Subtyping function $\mathcal{A}_2$).
input: *$\sigma \leqslant \tau$, where $\sigma \not\equiv \omega$ and $\tau \not\equiv \omega$ are ANFs;* ***output:*** *boolean.*

- *Case $\omega \leqslant \phi$: reject;*
- *Case $\omega \leqslant \sigma \to \tau$: reject;*
- *Case $\phi \leqslant \phi'$: if $\phi \equiv \phi'$ then accept, else reject;*
- *Case $\phi \leqslant \sigma \to \tau$: reject;*
- *Case $\sigma \to \tau \leqslant \phi$: reject;*
- *Case $\sigma \to \tau \leqslant \sigma' \to \tau'$: if $\mathcal{A}_1(\sigma' \leqslant \sigma)$ and $\mathcal{A}_1(\tau \leqslant \tau')$, then accept, else reject.*

The following two lemmas will be used to prove soundness and completeness of the algorithm $\mathcal{A}_1$.

Lemma 16.
1. *$\sigma \cup \tau \leqslant \rho$ iff $\sigma \leqslant \rho$ and $\tau \leqslant \rho$;*
2. *$\sigma \leqslant \tau \cap \rho$ iff $\sigma \leqslant \tau$ and $\sigma \leqslant \rho$.*

Proof. The two parts can be proved by examining the subtyping rules of the (sub)type theory Ξ. □

Lemma 17.
If all the σ_i and τ_j are ANFs, then:
1. *If $\exists j, \cap_i \sigma_i \leqslant \tau_j$, then $\cap_i \sigma_i \leqslant \cup_j \tau_j$;*
2. *If $\exists i, \sigma_i \leqslant \cup \tau_j$, then $\cap_i \sigma_i \leqslant \cup_j \tau_j$.*

Proof. The two parts can be proved by induction on the subtyping rules of the (sub)type theory Ξ using the ANF definition. □

The soundness proof is now straightforward.

Theorem 18 ($\mathcal{A}_1, \mathcal{A}_2$'s Soundness).
1. *Let σ (resp. τ) be in DANF (resp. CANF). If $\mathcal{A}_1(\sigma \leqslant \tau)$, then $\sigma \leqslant \tau$;*
2. *Let σ and τ be in ANF, such that $\tau \not\equiv \omega$. If $\mathcal{A}_2(\sigma \leqslant \tau)$, then $\sigma \leqslant \tau$.*

Proof. The proof follows the algorithm, therefore it proceeds by mutual induction.
1. *By case analysis on the algorithm $\mathcal{A}_1$ using Lemmas 16 and 17, and part 2;*
2. *By case analysis on the algorithm $\mathcal{A}_2$, and by looking at the subtyping rules.* □

Theorem 19 ($\mathcal{A}_1, \mathcal{A}_2$'s Completeness).
1. *For any type σ', τ' such that $\sigma' \leqslant \tau'$, let $\cup_i(\cap_j \sigma_{i,j}) \equiv \mathcal{R}_3 \circ \mathcal{R}_4 \circ \mathcal{R}_1(\sigma')$ and $\cap_h(\cup_k \tau_{h,k}) \equiv \mathcal{R}_2 \circ \mathcal{R}_4 \circ \mathcal{R}_1(\tau')$. We have that $\mathcal{A}_1(\cup_i(\cap_j \sigma_{i,j}) \leqslant \cap_h(\cup_k \tau_{h,k}))$;*
2. *Let σ and τ be in ANF, such that $\tau \not\sim \omega$. If $\sigma \leqslant \tau$, then $\mathcal{A}_2(\mathcal{R}_1(\sigma) \leqslant \mathcal{R}_1(\tau))$.*

Proof. We know by Lemma 12 that rewriting preserves subtyping, therefore as $\sigma' \leqslant \tau'$, we know that $\cup_i(\cap_j \sigma_{i,j}) \leqslant \cup_j \cap_h (\cup_k \tau_{h,k})$. The proof proceeds by mutual induction.
1. *The proof of this point relies on Lemmas 16 and 17: it is not shown by lack of space (see [LS17]) ;*
2.
   - *Case $\omega \leqslant \tau$: by hypothesis, $\omega \not\leqslant \tau$, so this case is absurd;*
   - *Case $\phi \leqslant \phi'$: we can show that $\phi \equiv \phi'$;*
   - *Case $\sigma \to \tau \leqslant \phi$: it can be proved that this case is absurd;*
   - *Case $\phi \leqslant \sigma \to \tau$: we can show that $\phi \leqslant \sigma \to \tau$ iff $\sigma \to \tau \sim \omega$, and this contradicts the hypothesis $\sigma \to \tau \not\sim \omega$: this is absurd;*
   - *Case $\sigma \to \tau \leqslant \sigma' \to \tau'$: we can show that $\tau \leqslant \tau'$, and $\sigma' \leqslant \sigma$. We conclude by induction hypothesis.* □

5 Conclusions

We mention some future research directions.

Completeness of $\mathcal{L}^{\cap\cup}$. We have not proven yet completeness for our logic towards $\mathsf{NJ}(\beta)$, but we conjecture that if G_Γ is a logical context and $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$, then $\Gamma \vdash M : \sigma$.

Strong/Relevant Implication is another proof-functional connective: as well explained in [BM94], it can be viewed as a special case of implication *"whose related function space is the simplest one, namely the one containing only the identity function"*. Relevant implication is well-known in the literature, corresponding to Meyer and Routley's Minimal Relevant Logic B^+ [MR72]. Following our parallelism between type systems for lambda-calculi *à la* Curry, *à la* Church, and logics, we could conjecture that strong implication, denoted by $\supset_r$ in the logic, by $\to_r$ in the type theory, and by λ_r in the typed lambda-calculus, can lead to the following type (assignment) rules, proof-functional logical inference, and Mints' realizer in $\mathsf{NJ}(\beta)$, respectively:

$$\frac{\Gamma \vdash \mathsf{I} : \sigma \to \tau}{\Gamma \vdash \mathsf{I} : \sigma \to_r \tau}\,(\to_r I) \qquad \frac{\Gamma, x{:}\sigma \vdash x@\Delta : \tau}{\Gamma \vdash \lambda x.x@\lambda_r x{:}\sigma.\Delta : \sigma \to_r \tau}\,(\to_r I)$$

$$\frac{\Gamma, x{:}\sigma \vdash \Delta : \tau \quad \wr\Delta\wr \equiv x}{\Gamma \vdash \lambda_r x{:}\sigma.\Delta : \sigma \to_r \tau}\,(\to_r I) \qquad \frac{G_\Gamma \vdash r_{\sigma\to\tau}[\mathsf{I}]}{G_\Gamma \vdash r_{\sigma\to_r\tau}[\mathsf{I}]}\,(\supset_r I)$$

As showed in Remark 8, even a stronger (sub)type theory of Ξ (*i.e.* the (sub)theory Π of [BDCd95]) cannot be overlapped with a sound and complete interpretation of (sub)types as (sub)sets. We conjecture that, by extending the proof-functional logic with relevant implication ($\mathcal{L}_{\leqslant}^{\cap\cup\to_r}$), we could achieve completeness, by combining explicit coercions and relevant abstractions, as the following derivation shows:

$$\cfrac{\cfrac{\cfrac{\Gamma \vdash x : \sigma \quad \sigma \leqslant \tau}{\Gamma \vdash (\tau)x : \tau} \quad \wr(\tau)x\wr \equiv x}{\Gamma \vdash \lambda_r x{:}\sigma.(\tau)x : \sigma \to_r \tau} \quad \Gamma \vdash \Delta : \sigma}{\Gamma \vdash (\lambda_r x{:}\sigma.(\tau)x)\,\Delta : \tau}$$

Dependent Types / Logical Frameworks. Our aim is to build a small logical framework *à la* Edinburgh Logical Framework [HHP93], featuring dependent types and proof-functional logical connectives. We conjecture that, in addition to the usual machinery dealing with dependent types and a suitable upgrade of the essence function, the following typing rules can be good candidates for a proof-functional LF extension:

$$\frac{\Gamma, x{:}\sigma \vdash \Delta : \tau \quad \wr\Delta\wr \equiv x}{\Gamma \vdash \lambda^r x{:}\sigma.\Delta : \Pi^r x{:}\sigma.\tau}\,(\Pi^r I) \quad \frac{\Gamma \vdash \Delta_1 : \sigma \quad \Gamma \vdash \Delta_2 : \tau \quad \wr\Delta_1\wr \equiv \wr\Delta_2\wr}{\Gamma \vdash \langle\Delta_1\,,\Delta_2\rangle : \sigma \cap \tau}\,(\cap I)$$

$$\frac{\begin{array}{c}\Gamma \vdash \Delta_1 : \Pi y{:}\sigma.\rho[\mathsf{in}_1^\tau\, y/x] \quad \wr\Delta_1\wr \equiv \wr\Delta_2\wr \\ \Gamma \vdash \Delta_2 : \Pi y{:}\tau.\rho[\mathsf{in}_2^\sigma\, y/x] \quad \Gamma \vdash \Delta_3 : \sigma \cup \tau\end{array}}{\Gamma \vdash [\Delta_1\,,\Delta_2]\,\Delta_3 : \rho[\Delta_3/x]}\,(\cup E)$$

Studying the behavior of proof-functional connectives would be beneficial to existing interactive theorem provers such as Coq or Isabelle, and dependently typed programming languages such as Agda, Beluga, Epigram, or Idris.

Prototype Implementation. We are currently implementing a small kernel for a logical framework featuring union and intersection types, as the $\Lambda_{t\leqslant}^{\cap\cup}$ calculus and the proof-functional logic $\mathcal{L}_{\leqslant}^{\cap\cup}$ does. The actual type system also features an experimental implementation of dependent-types *à la* LF following the above type rules, and of a *Read-Eval-Print-Loop* (REPL). We will put our future efforts to integrate our algorithm $\mathcal{A}$ to the type checker engine. We conjecture that our subtyping algorithm could be rewritten nondeterministically for an alternating Turing machine in polynomial time: this would mean that this problem is in PSPACE. This could be coherent with the fact that inclusion problem for regular tree languages is PSPACE-complete [Sei90]. The aim 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 can “squeeze” the proof-functional 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. Existing proof assistants could get some benefit if extended with a proof-functional logic. We are also started an encoding of the proof-functional operators of intersection and union in Coq. The actual state of the prototype can be retrieved at `https://github.com/cstolze/Bull`.

Acknowledgment. We are grateful to Ugo de'Liguoro, Daniel Dougherty, and the anonymous referees for their useful comments and suggestions.

References

Aik99. Alexander Aiken. Introduction to set constraint-based program analysis. *Sci. Comput. Program.*, 35(2):79–111, 1999.

AW93. Alexander Aiken and Edward L. Wimmers. Type inclusion constraints and type inference. In *FPCA*, pages 31–41. ACM, 1993.

Bar84. Henk P. Barendregt. *The λ-Calculus*. Studies in logic and the foundations of mathematics, North-Holland, 1984.

Bar13. Henk P. Barendregt. *The λ-Calculus with Types*. Association for Symbolic Logic, Cambridge University Press, 2013.

BCDC83. Henk P. Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A Filter Lambda Model and the Completeness of Type Assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

BCL92. Kim B. Bruce, Roberto Di Cosmo, and Giuseppe Longo. Provable isomorphisms of types. *Mathematical Structures in Computer Science*, 2(2):231–247, 1992.

BDCd95. Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo de'Liguoro. Intersection and union types: syntax and semantics. *Inf. Comput.*, 119(2):202–230, 1995.

BM94. Franco Barbanera and Simone Martini. Proof-functional connectives and realizability. *Archive for Mathematical Logic*, 33:189–211, 1994.

Dam94. Flemming M. Damm. Subtyping with union types, intersection types and recursive types. In *TACS*, pages 687–706, 1994.

DCGV97. Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, and Betti Venneri. The “relevance” of intersection and union types. *Notre Dame Journal of Formal Logic*, 38(2):246–269, 1997.

DdLS16. Daniel J. Dougherty, Ugo de'Liguoro, Luigi Liquori, and Claude Stolze. A realizability interpretation for intersection and union types. In *APLAS*, volume 10017 of *Lecture Notes in Computer Science*, pages 187–205. Springer, 2016.

DL10. Daniel J. Dougherty and Luigi Liquori. Logic and computation in a lambda calculus with intersection and union types. In *LPAR*, volume 6355 of *Lecture Notes in Computer Science*, pages 173–191. Springer, 2010.

DP04. Joshua Dunfield and Frank Pfenning. Tridirectional typechecking. In *POPL*, pages 281–292, 2004.

Dun14. Joshua Dunfield. Elaborating intersection and union types. *J. Funct. Program.*, 24(2-3):133–165, 2014.

FCB08. Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types. *J. ACM*, 55(4):19:1–19:64, 2008.

HHP93. Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, 1993.

Hin82. J. Roger Hindley. The simple semantics for Coppo-Dezani-Sallé types. In *International Symposium on Programming*, pages 212–226, 1982.

How80. William A. Howard. The Formulae-as-Types Notion of Construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic press, 1980.

LE85. Edgar G. K. Lopez-Escobar. Proof functional connectives. In *Methods in Mathematical Logic*, volume 1130 of *Lecture Notes in Mathematics*, pages 208–221. Springer-Verlag, 1985.

LR07. Luigi Liquori and Simona Ronchi Della Rocca. Intersection typed system *à la* Church. *Information and Computation*, 9(205):1371–1386, 2007.

LS17. Luigi Liquori and Claude Stolze. A Decidable Subtyping Logic for Intersection and Union Types. Research report, Inria, `https://hal.inria.fr/hal-01488428`, 2017.

Min89. Grigori Mints. The completeness of provable realizability. *Notre Dame Journal of Formal Logic*, 30(3):420–441, 1989.

MPS86. David B. MacQueen, Gordon D. Plotkin, and Ravi Sethi. An ideal model for recursive polymorphic types. *Information and Control*, 71(1/2):95–130, 1986.

MR72. Robert K. Meyer and Richard Routley. Algebraic analysis of entailment I. *Logique et Analyse*, 15:407–428, 1972.

Pfe93. Frank Pfenning. Refinement types for logical frameworks. In *TYPES*, pages 285–299, 1993.

Pie91. Benjamin C. Pierce. *Programming with intersection types, union types, and bounded polymorphism*. PhD thesis, Technical Report CMU-CS-91-205, Carnegie Mellon University, 1991.

Pot80. Garrel 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.

Pra65. Dag Prawitz. *Natural deduction: a proof-theoretical study*. PhD thesis, Almqvist & Wiksell, 1965.

Rey96. John C. Reynolds. Design of the programming language Forsythe. Technical Report CMU–CS–96–146, Carnegie Mellon University, 1996.

Sei90. Helmut Seidl. Deciding equivalence of finite tree automata. *Journal of Symbolic Logic*, 19(3):424–437, 1990.