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. *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 to classical *truth-functional* connectives where the meaning of a compound formula is dependent only on the truth value of its subformulas.
Using the proof-as-types and terms-as-propositions paradigms, we extend the proof-functional logic previously defined by the authors with a decidable subtyping relation and we show the extension to be isomorphic to the Barbanera-Dezani-de'Liguoro type assignment system: we also provide a sound interpretation of the proof-functional logic using *Mints' realizers*.
We present a decidable algorithm for subtyping in presence of intersection and union types. The algorithm is presented in declarative style, and it is conceived to work for the minimal (sub)type theory Ξ of the above type assignment system: it is proved to be sound and complete.

Keywords. Logics and lambda calculus, type and subtype systems, subtyping algorithm.

1 Introduction

This paper is a contribution to the study of *subtyping* in presence of *intersection* and *union* types and the role of such type system in *logical* investigations; it is a natural follow up of the 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, and Hindley [Hin84] gives a useful introduction and bibliography. 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. 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 dissertation combines also

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

unions and intersections [Pie91]; a recent implementation of a typed programming language featuring intersection and union type is [Dun14].

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 evidence 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 (*a.k.a.* intersection types) 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 to the question of realizing other "proof-functional" connectives, like *strong implication*, or Lopez-Escobar's *strong equivalence* or *provable type isomorphism* of Bruce, Di Cosmo and Longo [BL85,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 evidence for both $A \supset C$ and $B \supset C$. Symmetrically to intersection, and extending the 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 union and intersection types: our interest is not only theoretical but also pragmatic since, in a dependent type setting, it opens the door to using those type operators in logical frameworks and proof-assistants. We also inspect the relationship between pure (*à la* Curry) and typed (*à la* Church) 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 subsumption rule and the type theory Ξ, as defined in [BDCd95];
- $\Lambda^{\cap\cup}_{\mathsf{t}\leqslant}$, an extension of the typed lambda-calculus with strong products and strong pairs $\Lambda^{\cap\cup}_{\mathsf{t}}$, as defined in [DL10], with subtyping and explicit coercions;

- $\mathcal{L}^{\cap\cup}_{\leqslant}$, an extension of the proof-functional logic $\mathcal{L}^{\cap\cup}$ of [DdLS16] with *ad hoc* predicates for subtyping;
- $\mathsf{NJ}(\beta)$, a natural deduction system for derivations in first-order intuitionistic logic with untyped lambda-terms [Pra65].

Judgements in these systems take the following four forms below:

$$\begin{array}{lllllll}
\mathbf{\Lambda}^{\cap\cup}_{\mathsf{u}\leqslant} & B, & x_\iota & : \tau & \vdash & M & : \sigma \\
\mathbf{\Lambda}^{\cap\cup}_{\mathsf{t}\leqslant} & \Gamma^{@}, & x_\iota \mathbin{@} \iota & : \tau & \vdash & M \mathbin{@} \Delta & : \sigma \\
\mathcal{L}^{\cap\cup}_{\leqslant} & \Gamma, & \iota & : \tau & \vdash & \Delta & : \sigma \\
\mathsf{NJ}(\beta) & G_B, & r_\tau[x_\iota] & & \vdash & r_\sigma[M] &
\end{array}$$

On the right-hand sides of the turnstiles, M is an untyped lambda-term, Δ is a simply-typed lambda-term with strong conjunction, strong disjunction, and explicit coercions, and σ is a simple type formed using $\to, \cap,$ and $\cup$. The $r_\sigma[M]$ are typing predicates to be realized. Note that B (resp. Γ) is obtained from $\Gamma^{@}$ by erasing all the $@\iota$ (resp. "$x@$"), and G_B is obtained by B by "realizing" all the $x_\iota{:}\tau$.

Our first contribution is 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 are a pure lambda-term, while Δ are lambda-terms enriched with strong-products and and strong-sums. 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, that is the fact that in an intersection (resp. union) the two Δ relate to the same M. Thanks to the erasing essence 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; we also show that the extended $\mathcal{L}^{\cap\cup}_{\leqslant}$ logic of subtyping is sound with respect to the realizability logic $\mathsf{NJ}(\beta)$.

Our second contribution is to present a decidable algorithm for subtyping in presence of intersection and union types. The algorithm is presented in "functional style", and it is conceived to work for the minimal (sub)type theory Ξ (*i.e.* axioms 1 to 14, as presented in [BDCd95]), the latter theory being compatible with a set-based interpretation of $\to, \cap, \cup, \leqslant$ with function space, set intersection, and set union, respectively.

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 assignments 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 in 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].

Barbanera, Dezani-Ciancaglini, and de'Liguoro [BDCd95] introduced an untyped lambda-calculus $\Lambda^{\cap\cup}_{\mathsf{u}\leqslant}$ with 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 original type assignment system (but without subtyping). In [DdLS16] we introduced the new notion of *essence* $\wr\Delta\wr$ of a typed lambda-term Δ, used to understand the connection between pure terms and typed terms. Intuitively, these three statements are equivalent:

- $\Gamma \overset{@}{\vdash} M@\Delta : \sigma$
- $\Gamma \vdash \Delta : \sigma$ and $\wr\Delta\wr =_\beta M$
- $B \vdash M : \sigma$

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 Ξ and Π of [BDCd95]: this motivates the use of the (sub)type theory Ξ with their natural correspondence with $\mathsf{NJ}(\beta)$.

Roger Hindley gave first a subtyping algorithm for type intersection [Hin82], and there is a rich literature reducing the subsetting problem in presence of set intersection and set union to set constraint-based problem: good references are [Aik99,Dam94,DP04]. 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, union, intersection, complementation and constructor types. Their algorithm rewrites the 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.

Frank Pfenning work on Refinement Types [Pfe93] pioneered an extension of Edinburgh Logical Framework with subtyping and intersection types: our

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

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

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

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, Δ, and the derived $M@\Delta$ are defined using the following three syntactic categories:

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

where ϕ denotes arbitrary constant types and ω denotes a special type that is inhabited by all terms. The Δ-expression $\langle \Delta\,, \Delta \rangle$ denotes the strong pair while $[\Delta\,, \Delta]$ denotes the strong sum, with the respective projections and injections, respectively. Finally $[\sigma]\Delta$ denotes the explicit coercion of Δ 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 residuals of redexes in M are contracted simultaneously. Reduction for the calculus *à la* Church $\Lambda^{\cap\cup}_{\mathsf{t}}$ is delicate because it must keep *synchronized* the untyped reduction of M with the typed reduction of Δ: it is defined in [DL10]. Reductions in $\mathcal{L}^{\cap\cup}$ is ordinary β-reduction plus the following four reduction rules:

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

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 $\Lambda^{\cap\cup}_{\mathsf{t}}$ of [DL10]: note that this type system is completely syntax directed.

On subject reduction in $\Lambda^{\cap\cup}_{\mathsf{u}\leqslant}$ and in $\mathcal{L}^{\cap\cup}$. As well explained in [BDCd95] the type assignment system $\Lambda^{\cap\cup}_{\mathsf{u}}$ is not invariant under β-reduction: invariance is

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

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

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

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

recovered under *parallel* β-reduction: Pierce's counter example [Pie91] shows the failure of subject reduction for simple, non parallel, β-reduction:

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

Under the type context $B \triangleq x{:}(\sigma_1 \to \sigma_1 \to \tau) \cap (\sigma_2 \to \sigma_2 \to \tau), y{:}\rho \to \sigma_1 \cup \sigma_2, z{:}\rho$, the first and the last terms can be typed with τ, while terms in the "fork" are not because of the mismatch of the premises in the $(\cup E)$ type assignment rule. The situation is quite different in $\mathcal{L}^{\cap\cup}$: under the type context $\Gamma \triangleq \iota_x{:}(\sigma_1 \to \sigma_1 \to \tau) \cap (\sigma_2 \to \sigma_2 \to \tau), \iota_y{:}\rho \to \sigma_1 \cup \sigma_2, \iota_z{:}\rho$, the corresponding Δ-term is:

$$[\underbrace{(\lambda\iota{:}\sigma_1.(\mathsf{pr}_1\,\iota_x)\,\iota\,\iota)}_{\Delta_1}, \underbrace{(\lambda\iota{:}\sigma_2.(\mathsf{pr}_2\,\iota_x)\,\iota\,\iota)}_{\Delta_2}]\,(\underbrace{(\lambda\iota{:}\rho \to \sigma_1 \cup \sigma_2.\iota)}_{\Delta_3}\,\iota_y\,\iota_z)$$

By inspecting the reduction rules of $\mathcal{L}^{\cap\cup}$ we can see that the only applicable redex is the β-redex $\Delta_3\,\iota_y$, and that gives $[\Delta_1\,, \Delta_2]\,(\iota_y\,\iota_z)$: this Δ-term is typable with τ and it will "correspond" to the untyped reductum $x\,(y\,z)\,(y\,z)$ in [BDCd95].

The next definition will clarify what we intend with "correspondance" 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 untyped λ-terms: intuitively two typed Δ-terms proves 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:

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

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

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

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

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

Figure 3 presents the main rules of the proof-functional logic $\mathcal{L}^{\cap\cup}$, as presented in [DdLS16]. The logic $\mathcal{L}^{\cap\cup}$, introduced in [DdLS16] is a *proof-functional*, in the sense of Pottinger [Pot80] and Lopez-Escobar [LE85]: formulas encode, using the Curry-Howard isomorphism, *derivations* $\mathcal{D} : B \vdash M : \sigma$ in the type assignment system $\Lambda_{\mathsf{u}}^{\cap\cup}$ which are, in turn, isomorphic to typed *judgments* $\Gamma \overset{@}{\vdash} M@\Delta : \sigma$ 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]. Subtyping is a preorder over types, and it is written as $\sigma \leqslant \tau$: we use the standard terminology of "(sub)type theories" for any collection of inequalities between types satisfying natural closure conditions. As such, 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) $\sigma \leqslant \sigma \cap \sigma$
(2) $\sigma \cup \sigma \leqslant \sigma$
(3) $\sigma \cap \tau \leqslant \sigma, \sigma \cap \tau \leqslant \tau$
(4) $\sigma \cup \tau \leqslant \sigma, \sigma \cup \tau \leqslant \tau$
(5) $\sigma \leqslant \omega$
(6) $\sigma \leqslant \sigma$
(7) $\sigma_1 \leqslant \sigma_2, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \cap \tau_1 \leqslant \sigma_2 \cap \tau_2$
(8) $\sigma_1 \leqslant \sigma_2, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \cup \tau_1 \leqslant \sigma_2 \cup \tau_2$
(9) $\sigma \leqslant \tau, \tau \leqslant \rho \Rightarrow \sigma \leqslant \rho$
(10) $\sigma \cap (\tau \cup \rho) \leqslant (\sigma \cap \tau) \cup (\sigma \cap \rho)$
(11) $(\sigma \to \tau) \cap (\sigma \to \rho) \leqslant \sigma \to (\tau \cap \rho)$
(12) $(\sigma \to \rho) \cap (\tau \to \rho) \leqslant (\sigma \cup \tau) \to \rho$
(13) $\omega \leqslant \omega \to \omega$
(14) $\sigma_2 \leqslant \sigma_1, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \to \tau_1 \leqslant \sigma_2 \to \tau_2$

The 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 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 or an explicit coercion rule, can be defined as follows:

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

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

The next theorem relate 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 $\wr\Delta\wr \equiv M$ and Γ (resp. B) is obtained by erasing all the $x@$ (resp. $@\iota$) in $\overset{@}{\Gamma}$.

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

Proof. Point 1,2 by using Theorem 10 of [DL10]; point 3 by 1,2.

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

Theorem 3 (Conservativity).
The typed system $\Lambda^{\cap\cup}_{\mathsf{t}\leqslant}$ and the proof-functional logics $\mathcal{L}^{\cap\cup}_{\leqslant}$, 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 $\Lambda^{\cap\cup}_{\mathsf{t}\leqslant}$), 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 $\Lambda^{\cap\cup}_{\mathsf{t}\leqslant}$ we proceeds by upgrading results in Section 5.4 of [DL10] with the type rule $(\leqslant)$. Properties of $\mathcal{L}^{\cap\cup}_{\leqslant}$ are mostly inherited by $\Lambda^{\cap\cup}_{\mathsf{t}\leqslant}$ or, as for case of subject reduction for β-, pr_i - and in_i -reductions, proved by induction on the structure of the derivation.

3 Realizers

We start this section by recalling the logic NJ. 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 $\mathsf{NJ}(\beta)$[1], extends NJ, with untyped lambda-terms and predicates $\mathbf{P}_\phi(x)$, the latter being axiomatized via the two Post rules:

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

[1] For commodity, the full system is shown in Figure 4 in the appendix for referees

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 $B \vdash \wr\Delta\wr : \sigma$ in $\Lambda^{\cap\cup}_{\mathsf{u}\leqslant}$. We define $G_B \triangleq r_{\sigma_1}[x_1], \ldots, r_{\sigma_n}[x_n]$ and $B \triangleq \{x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n\}$. For a given context $\Gamma \triangleq \{\iota_{x_1}{:}\sigma_1, \ldots, \iota_{x_n}{:}\sigma_n\}$ in $\mathcal{L}^{\cap\cup}_{\leqslant}$, we associate a logical context $G_\Gamma \triangleq r_{\sigma_1}[x_1], \ldots, r_{\sigma_n}[x_n]$. Note also that $G_{\Gamma,\iota_x:\sigma} \equiv G_\Gamma, r_\sigma[x]$ and $x \notin \mathsf{Fv}(G_\Gamma)$, since $\iota_x \notin Dom(\Gamma)$, by context definition. 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 $B \vdash M : \sigma$ then $G_B \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$.

Proof. The complete proof given in the appendix for referees.

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 Theorem 6, we can now state soudness:

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. Using Theorem 2 part 3 we can prove that if $B \vdash M : \sigma$ then $G_B \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$.

- *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 the 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:*

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

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 $\wr\Delta\wr =_{\beta\eta} M$ is more tricky because of the presence of the union elimination rule $(\vee E)$ in $\mathsf{NJ}(\beta)$. As an example, let $\phi \triangleq (\sigma\cup\tau)\cap(\sigma\cup\rho) \to \sigma\cup(\tau\cap\rho)$: 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

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

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 inhabits $\{a, b\} \to \{a\}$ 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 and complete *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 $\Lambda^{\cap\cup}_{\mathsf{t}\leqslant}$. This section presents a decidable algorithm $\mathcal{A}$ for subtyping in the (sub)type theory Ξ: The algorithm $\mathcal{A}$ needs four decidable subroutines, all implemented using term rewriting systems:

– $\mathcal{R}_1$, 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.

Algorithm $\mathcal{A}$ is polynomial, and, as it usually is the case when putting formulas in normal form, the preprocessing of $\sigma \leqslant \tau$ using the subroutines $\mathcal{R}_{1,2,3,4}$ is exponential.

Our algorithm $\mathcal{A}$ is proved to be sound and complete *w.r.t.* 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.

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* conjunctive normal form *(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* disjunctive normal form *(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 $(\sigma \cap \rho) \cup (\tau \cup \rho)$.*

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.

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 $\mathcal{R}_3(\sigma) \to \mathcal{R}_2(\tau)$;*
  - *$\cup_i \sigma_i \to \cap_h \tau_h$ rewrites to $\cap_i(\cap_h(\sigma_i \to \tau_h))$.*

Since $\mathcal{R}_2$ and $\mathcal{R}_3$ terminate, $\mathcal{R}_4$ terminates and its complexity is exponential.

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.

The next definition introduce the "preprocessing" of types necessary to feed correctly the algorithm $\mathcal{A}$.

Definition 13.

- *A type is in* conjunctive-arrow normal form *(CANF) if it is in CNF and all the arrow type subterms are in ANF. The composition of the term rewriting systems $\mathcal{R}_2 \circ \mathcal{R}_4 \circ \mathcal{R}_1$ rewrite a type into its CANF;*
- *A type is in* disjunctive-arrow normal form *(DANF) if it is in DNF and all the arrow type subterms are in ANF. The composition of the term rewriting systems $\mathcal{R}_3 \circ \mathcal{R}_4 \circ \mathcal{R}_1$ rewrite a type into its DANF.*

4.1 The algorithm $\mathcal{A}$

Our algorithm $\mathcal{A}$ accepts or rejects an $\sigma \leqslant \tau$ formula: in a nutshell the algorithm $\mathcal{A}$ proceeds as follows: using $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$ and $\mathcal{R}_4$, we first "preprocess" σ into a DANF and τ into a CANF producing a (normalized) subtyping statement of the shape $\cup_i(\cap_j \sigma_{i,j}) \leqslant \cap_h(\cup_k \tau_{h,k})$, where all the $\sigma_{i,j}, \tau_{h,k}$ are in ANF; then we call $\mathcal{A}_1$. More precisely, $\mathcal{A}$ is composed by two mutually inductive functions, called $\mathcal{A}_1$ and $\mathcal{A}_2$.

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:*** *bool.*

- *if $\cap_h(\cup_k \tau_{h,k})$ is ω, then accept;*
- *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 σ and $\tau \not\equiv \omega$ are ANFs;* ***output:*** *bool.*

- *Case $\omega \leqslant \phi$: reject;*
- *Case $\omega \leqslant \sigma \to \tau$: reject;*

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

The following two technical lemmas are useful 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.*

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* $\tau \not\equiv \omega$ *be in ANF. If* $\mathcal{A}_2(\sigma \leqslant \tau)$ *then* $\sigma \leqslant \tau$*.*

Proof. The proof proceeds by mutual induction, where base case is in proving part (ii)*.*

1. *By case analysis on the algorithm* $\mathcal{A}_1$ *using Lemmas 16 and 17 and part (ii);*
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* $\tau \not\sim \omega$ *be ANFs. If* $\sigma \leqslant \tau$ *then* $\mathcal{A}_2(\mathcal{R}_1(\sigma) \leqslant \mathcal{R}_1(\tau))$*.*

Proof. See Theorem 29 in the appendix for referees.

5 Conclusions

We mention some future research directions.

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:

$$\frac{B \vdash \mathsf{I} : \sigma \to \tau}{B \vdash \mathsf{I} : \sigma \to_r \tau} \quad \frac{\Gamma^{@}, x_\iota @ \iota{:}\sigma \vdash x_\iota @ \Delta : \tau}{\Gamma^{@} \vdash \lambda x_\iota . x_\iota @ \lambda_r \iota{:}\sigma . \Delta : \sigma \to_r \tau} \quad \frac{\Gamma, \iota{:}\sigma \vdash \Delta : \tau \quad \wr \Delta \wr =_\beta \iota}{\Gamma \vdash \lambda_r \iota{:}\sigma . \Delta : \sigma \to_r \tau} \quad \frac{G_B \vdash r_{\sigma\to\tau}[\mathsf{I}]}{G_B \vdash r_{\sigma\to_r\tau}[\mathsf{I}]}$$

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

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \iota : \sigma \quad \sigma \leqslant \tau}{\Gamma \vdash (\tau)\iota : \tau} \qquad \wr(\tau)\iota\wr =_\beta \iota}{\Gamma \vdash \lambda_r \iota{:}\sigma . (\tau)\iota : \sigma \to_r \tau} \qquad \Gamma \vdash \Delta : \sigma}{\Gamma \vdash (\lambda_r \iota{:}\sigma . (\tau)\iota)\, \Delta : \tau}$$

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 compatible with a set-based interpretation of $\to, \cap, \cup$ with function space, set intersection, and set union, respectively. We conjecture that, in addition to the usual machinery dealing with dependent types, the following typing rules can be good candidates for a proof-functional LF extension:

$$\frac{\Gamma, \iota{:}\sigma \vdash \Delta : \tau \quad \wr\Delta\wr \equiv x_i}{\Gamma \vdash \lambda^r \iota{:}\sigma . \Delta : \Pi^r \iota{:}\sigma . \tau}\ (\Pi^r I) \qquad \frac{\Gamma \vdash \Delta_1 : \sigma_1 \quad \Gamma \vdash \Delta_2 : \sigma_2 \quad \wr\Delta_1\wr \equiv \wr\Delta_2\wr}{\Gamma \vdash \Delta_1 \cap \Delta_2 : \sigma_1 \cap \sigma_2}\ (\cap I)$$

$$\frac{\begin{array}{l}\Gamma \vdash \Delta_1 : \Pi \iota' {:}\sigma_1 . \sigma_3[\mathsf{in}_1\, \iota'/\iota] \quad \wr\Delta_1\wr \equiv \wr\Delta_2\wr \\ \Gamma \vdash \Delta_2 : \Pi \iota' {:}\sigma_2 . \sigma_3[\mathsf{in}_2\, \iota'/\iota] \quad \Gamma \vdash \Delta_3 : \sigma_1 \cup \sigma_2\end{array}}{\Gamma \vdash (\Delta_1 \cup \Delta_2)\, \Delta_3 : \sigma_3[\Delta_3/\iota]}\ (\cup E)$$

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 $\Lambda^{\cap\cup}_{\mathsf{t}\leqslant}$ calculus and the proof-functional logic $\mathcal{L}^{\cap\cup}_{\leqslant}$ 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 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 and Daniel Dougherty for the many useful 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.

BCDC83. Henk 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.

BL85. Kim B. Bruce and Giuseppe Longo. Provable isomorphisms and domain equations in models of typed languages (preliminary version). In *Proceedings of STOC*, pages 263–272, 1985.

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.

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 coppe-dezani-sallé types. In *International Symposium on Programming*, pages 212–226, 1982.

Hin84. J. Roger Hindley. Coppo-Dezani types do not correspond to propositional logic. *Theor. Comput. Sci.*, 28:235–236, 1984.

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.

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. Report CMU–CS–96–146, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1996.

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

A Appendix (for Referees)

$$\frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \bot}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A}\ (\bot) \qquad \frac{A \in G_\Gamma}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A}\ (Hyp)$$

$$\frac{G_\Gamma, A \vdash_{\mathsf{NJ}(\beta)} B}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A \supset B}\ (\supset I) \qquad \frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A \supset B \quad G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} B}\ (\supset E)$$

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

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

$$\frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \forall x.A}\ (\forall I) \qquad \frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \forall x.A}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A[M/x]}\ (\forall E)$$

$$\frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A[M/x]}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \exists x.A}\ (\exists I) \qquad \frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \exists x.A \quad G_\Gamma, A \vdash_{\mathsf{NJ}(\beta)} B}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} B}\ (\exists E)$$

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

Fig. 4. The Logic $\mathsf{NJ}(\beta)$

The next definition is reminiscent to filters, as seen in [BCDC83]: more precisely $L^{\geqslant}_\sigma$ is the smallest filter containing σ *w.r.t.* Definition 2.6. of [BCDC83].

Definition 20. ***(Filters)***

1. *$L^{\geqslant}_\omega$ is the least set such that:*
   - $\omega \in L^{\geqslant}_\omega$;
   - $\forall \sigma, \tau \in L^{\geqslant}_\omega$ *we have that* $\sigma \cap \tau, \tau \cap \sigma \in L^{\geqslant}_\omega$;
   - $\forall \sigma \in L^{\geqslant}_\omega, \forall \tau$ *we have that* $\sigma \cup \tau, \tau \cup \sigma \in L^{\geqslant}_\omega$;
   - $\forall \tau \in L^{\geqslant}_\omega, \forall \sigma$ *we have that* $\sigma \to \tau \in L^{\geqslant}_\omega$.
2. *For any atomic type ϕ, $L^{\geqslant}_\phi$ is the least set such that:*
   - $\phi \in L^{\geqslant}_\phi$;
   - $\forall \sigma, \tau \in L^{\geqslant}_\phi$ *we have that* $\sigma \cap \tau, \tau \cap \sigma \in L^{\geqslant}_\phi$;
   - $\forall \sigma \in L^{\geqslant}_\phi, \forall \tau$ *we have that* $\sigma \cup \tau, \tau \cup \sigma \in L^{\geqslant}_\phi$;
   - $\forall \sigma \in L^{\geqslant}_\omega$ *we have that* $\sigma \in L^{\geqslant}_\phi$.
3. *For any arrow type $\sigma \to \tau$, $L^{\geqslant}_{\sigma\to\tau}$ is the least set such that:*
   - $\forall \sigma', \tau'$ *such that* $\sigma' \leqslant \sigma$ *and* $\tau \leqslant \tau'$, *we have that* $\sigma' \to \tau' \in L^{\geqslant}_{\sigma\to\tau}$;
   - $\forall \sigma', \tau' \in L^{\geqslant}_{\sigma\to\tau}$ *we have that* $\sigma' \cap \tau', \sigma' \cap \tau' \in L^{\geqslant}_{\sigma\to\tau}$;
   - $\forall \sigma' \in L^{\geqslant}_{\sigma\to\tau}, \forall \tau'$ *we have that* $\sigma' \cup \tau', \tau' \cup \sigma' \in L^{\geqslant}_{\sigma\to\tau}$;

   - $\forall \sigma' \in L^{\geqslant}_{\omega}$ *we have that* $\sigma' \in L^{\geqslant}_{\sigma\to\tau}$.
4. *For any intersection of types* $\cap_i\sigma_i$ *such that the* σ_i *are either atomic types, arrow types, or* ω, $L^{\geqslant}_{\cap_i\sigma_i}$ *is the least set such that:*
   - $\forall i, L^{\geqslant}_{\sigma_i} \subseteq L^{\geqslant}_{\cap_i\sigma_i}$;
   - $\forall \sigma', \tau' \in L^{\geqslant}_{\cap_i\sigma_i}$ *we have that* $\sigma' \cap \tau', \sigma' \cap \tau' \in L^{\geqslant}_{\cap_i\sigma_i}$;
   - $\forall \sigma' \in L^{\geqslant}_{\cap_i\sigma_i}, \forall \tau'$ *we have that* $\sigma' \cup \tau', \tau' \cup \sigma' \in L^{\geqslant}_{\cap_i\sigma_i}$;
   - $\forall \sigma\to\tau, \sigma\to\rho \in L^{\geqslant}_{\cap_i\sigma_i}$ *we have that* $L^{\geqslant}_{\sigma\to(\tau\cap\rho)} \subseteq L^{\geqslant}_{\cap_i\sigma_i}$;
   - $\forall \sigma\to\rho, \tau\to\rho \in L^{\geqslant}_{\cap_i\sigma_i}$ *we have that* $L^{\geqslant}_{(\sigma\cup\tau)\to\tau} \subseteq L^{\geqslant}_{\cap_i\sigma_i}$.

The next definition introduces *co-filters*, *i.e.* $L^{\leqslant}_{\sigma}$ is the smallest set containing all the types smaller than σ.

Definition 21. ***(Co-Filters)***

1. $L^{\leqslant}_{\omega}$ *is the set of all types;*
2. *For any atomic type* ϕ, $L^{\leqslant}_{\phi}$ *is the least set such that:*
   - $\phi \in L^{\leqslant}_{\phi}$;
   - $\forall \sigma \in L^{\leqslant}_{\phi}, \forall \tau$ *we have that* $\sigma \cap \tau, \tau \cap \sigma \in L^{\leqslant}_{\phi}$;
   - $\forall \sigma, \tau \in L^{\leqslant}_{\phi}$ *we have that* $\sigma \cup \tau, \tau \cup \sigma \in L^{\leqslant}_{\phi}$.
3. *For any* σ, τ *such that* $\omega \leqslant \tau$ *(i.e.* $\omega \leqslant \sigma\to\tau$*),* $L^{\leqslant}_{\sigma\to\tau}$ *is the set of all types;*
4. *For any* $\cap_i\sigma_i, \cup_j\tau_j, \cup_k\rho_k$ *such that all the* σ_i, τ_j, ρ_k *are ANFs and* $\omega \not\leqslant \cup_j\tau_j$*, we define by mutual induction the sets* $L^{\leqslant}_{\cap_i\sigma_i\to\cup_j\tau_j}$ *and* $L^{\leqslant}_{\cup_k\rho_k}$*:*
   - $\forall k, L^{\leqslant}_{\rho_k} \subseteq L^{\leqslant}_{\cup_k\rho_k}$;
   - $\forall \sigma \in L^{\leqslant}_{\cup_k\rho_k}, \forall \tau$ *we have that* $\sigma \cap \tau, \tau \cap \sigma \in L^{\leqslant}_{\cup_k\rho_k}$;
   - $\forall \sigma, \tau \in L^{\leqslant}_{\cup_k\rho_k}$ *we have that* $\sigma \cup \tau, \tau \cup \sigma \in L^{\leqslant}_{\cup_k\rho_k}$;
   - $\forall \sigma \in L^{\geqslant}_{\cap_i\sigma_i}, \forall \tau \in L^{\leqslant}_{\cup_j\tau_j}$ *we have that* $\sigma\to\tau \in L^{\leqslant}_{\cap_i\sigma_i\to\cup_j\tau_j}$;
   - $\forall \sigma \in L^{\leqslant}_{\cap_i\sigma_i\to\cup_j\tau_j}, \forall \tau$ *we have that* $\sigma \cap \tau, \tau \cap \sigma \in L^{\leqslant}_{\cap_i\sigma_i\to\cup_j\tau_j}$;
   - $\forall \sigma, \tau \in L^{\leqslant}_{\cap_i\sigma_i\to\cup_j\tau_j}$ *we have that* $\sigma \cup \tau, \tau \cup \sigma \in L^{\leqslant}_{\cap_i\sigma_i\to\cup_j\tau_j}$.

The next lemma proves that $L^{\geqslant}_{\sigma}$ are [BCDC83] filters.

Lemma 22.
For any σ *such that* $L^{\geqslant}_{\sigma}$ *is defined, we have:*

1. $\tau \in L^{\geqslant}_{\sigma}$ *iff* $\sigma \leqslant \tau$;
2. $\sigma \leqslant \tau \cup \rho$ *iff* $\sigma \leqslant \tau$ *or* $\sigma \leqslant \rho$.

Proof. Since (ii) is deducible from (i) by looking at the construction rules of $L^{\geqslant}_{\sigma}$*, we only have to prove (i), as follows.*

First we prove that, if $\tau \in L^{\geqslant}_{\sigma}$ *then* $\sigma \leqslant \tau$*: by induction on the construction rules of* $L^{\geqslant}_{\omega}$*, for any* $\sigma \in L^{\geqslant}_{\omega}$*, we have that* $\omega \leqslant \sigma$*. For the same reasons, if* $\sigma \in L^{\geqslant}_{\phi}$*, we have that* $\phi \leqslant \sigma$*; if* $\sigma' \in L^{\geqslant}_{\sigma\to\tau}$*, we have that* $\sigma \to \tau \leqslant \sigma'$*; if* $\sigma \in L^{\geqslant}_{\cup_i\sigma_i}$*, we have that* $\cup_i\sigma_i \leqslant \sigma$.

Now we prove that, if $\sigma \leqslant \tau$*, then* $\tau \in L^{\geqslant}_{\sigma}$*:*

- *Case ω: by induction on the subtyping rules of the (sub)type theory Ξ, we show that for any σ, τ such that $\sigma \leqslant \tau$ and $\sigma \in L^{\geqslant}_{\omega}$, we have that $\tau \in L^{\geqslant}_{\omega}$. Some cases of the inductive proof need a case analysis on the construction rules of $L^{\geqslant}_{\omega}$. Therefore, if $\omega \leqslant \tau$, then $\tau \in L^{\geqslant}_{\omega}$;*
- *Case ϕ: for any σ, τ such that $\sigma \leqslant \tau$ and $\sigma \in L^{\geqslant}_{\phi}$, we have two cases:*
  - *If $\omega \leqslant \sigma$, then, as $L^{\geqslant}_{\omega} \subseteq L^{\geqslant}_{\phi}$, we know that $\tau \in L^{\geqslant}_{\omega}$, so it is clear that $\tau \in L^{\geqslant}_{\phi}$;*
  - *If $\omega \not\leqslant \sigma$, then we can prove by induction on the subtyping rules of the (sub)type theory Ξ that $\tau \in L^{\geqslant}_{\omega}$.*

  Therefore, if $\phi \leqslant \tau$, then $\tau \in L^{\geqslant}_{\phi}$;
- *Case $\sigma \to \tau$: for any σ', τ' such that $\sigma' \leqslant \tau'$ and $\sigma' \in L^{\geqslant}_{\sigma \to \tau}$, we have two cases:*
  - *If $\omega \leqslant \sigma'$, then it is clear that $\tau' \in L^{\geqslant}_{\sigma \to \tau}$;*
  - *If $\omega \not\leqslant \sigma'$, then we can prove by induction on the subtyping rules of the (sub)type theory Ξ that $\tau' \in L^{\geqslant}_{\sigma \to \tau}$. The most difficult cases are rules (12) and (13) of type theory Ξ. In rule (12), we suppose $\sigma' \equiv (\sigma_1 \to \tau_1) \cap (\sigma_1 \to \tau_2) \in L^{\geqslant}_{\sigma \to \tau}$. From the construction rules of $L^{\geqslant}_{\sigma \to \tau}$, we know that both $\sigma_1 \to \tau_1 \in L^{\geqslant}_{\sigma \to \tau}$ and $\sigma_1 \to \tau_2 \in L^{\geqslant}_{\sigma \to \tau}$. These types cannot be both in $L^{\geqslant}_{\omega}$, because of the hypothesis $\omega \not\leqslant \sigma'$. From the construction rules of $L^{\geqslant}_{\sigma \to \tau}$, we know that both $\tau \leqslant \tau_1$ and $\tau \leqslant \tau_2$, so by Lemma 16 we get $\tau \leqslant \tau_1 \cap \tau_2$. For the same reason, we get $\sigma_1 \leqslant \sigma$. Therefore $\sigma_1 \to (\tau_1 \cap \tau_2) \in L^{\geqslant}_{\sigma \to \tau}$. Proof for rule (13) follows the same path.*

  As $\sigma \to \tau \in L^{\geqslant}_{\sigma \to \tau}$, we conclude that, if $\sigma \to \tau \leqslant \tau'$, then $\tau' \in L^{\geqslant}_{\sigma \to \tau}$.
- *Case $\cap_i \sigma_i$: we prove by induction on the subtyping rules of the (sub)type theory Ξ that for any σ, τ such that $\sigma \leqslant \tau$ and $\sigma \in L^{\geqslant}_{\cap_i \sigma_i}$, we have that $\tau \in L^{\geqslant}_{\cap_i \sigma_i}$. As $\cap_i \sigma_i \in L^{\geqslant}_{\cap_i \sigma_i}$, we conclude that, if $\cap_i \sigma_i \leqslant \tau$, then $\tau \in L^{\geqslant}_{\cap_i \sigma_i}$.*

The next corollary proves the reciprocate of rule (14) of the (sub)type theory Ξ. It will be crucial to prove the completeness of $\mathcal{A}_2$.

Corollary 23. *If $\sigma \to \tau \leqslant \sigma' \to \tau'$ and $\omega \not\leqslant \tau'$, then $\sigma' \leqslant \sigma$ and $\tau \leqslant \tau'$.*

Proof. According to the construction rules of $L^{\geqslant}_{\omega}$, we have that $\sigma' \to \tau' \notin L^{\geqslant}_{\omega}$. Therefore, by looking at the definition of $L^{\geqslant}_{\sigma \to \tau}$, we have that $\sigma' \leqslant \sigma$ and $\tau \leqslant \tau'$.

The next lemma proves that the $\mathcal{R}_1$ preprocessing allows us to decide whether a type is equivalent to ω.

Lemma 24. *If $\sigma \sim \omega$, then $\mathcal{R}_1$ rewrites σ as ω.*

Proof. Since $\sigma \sim \omega$, we have that $\sigma \in L^{\geqslant}_{\omega}$. The proof proceeds by induction on σ.

The next lemma proves that $L^{\leqslant}_{\sigma}$ are co-filters.

Lemma 25.
For any τ such that $L^{\leqslant}_{\tau}$ is defined, we have:

1. $\sigma \in L^{\leqslant}_{\tau}$ *iff* $\sigma \leqslant \tau$;
2. $\sigma \cap \rho \leqslant \tau$ *iff* $\sigma \leqslant \tau$ *or* $\rho \leqslant \tau$.

Proof. Since (ii) is deducible from (i) by looking at the construction rules of $L^{\leqslant}_{\sigma}$, we only have to prove (i), as follows.

First we prove that, if $\sigma \in L^{\leqslant}_{\tau}$ then $\sigma \leqslant \tau$: by induction on the construction rules of $L^{\leqslant}_{\phi}$, if $\sigma \in L^{\leqslant}_{\phi}$, we have that $\sigma \leqslant \phi$. For the same reasons, if $\sigma \in L^{\leqslant}_{\cap_i \sigma_i \to \cup_j \tau_j}$, we have that $\sigma \leqslant \cap_i \sigma_i \to \cup_j \tau_j$, and if $\sigma \in L^{\leqslant}_{\cup_i \sigma_i}$, we have that $\sigma \leqslant \cup_i \sigma_i$.

Now we prove that, if $\sigma \leqslant \tau$, then $\sigma \in L^{\leqslant}_{\tau}$:

- Case ω: for $L^{\leqslant}_{\omega}$ and $L^{\leqslant}_{\sigma \to \tau}$ where $\omega \leqslant \tau$, the proof is trivial;
- Case ϕ: we prove by induction on the subtyping rules of the (sub)type theory Ξ that for any σ, τ such that $\tau \leqslant \sigma$ and $\tau \in L^{\leqslant}_{\phi}$, we have that $\sigma \in L^{\leqslant}_{\phi}$. Therefore, if $\sigma \leqslant \phi$, then $\sigma \in L^{\leqslant}_{\phi}$;
- Cases $\cup_k \rho_k$ and $\cap_i \sigma_i \to \cup_j \tau_j$: this is the most difficult case. We notice that $\cup_k \rho_k \in L^{\leqslant}_{\cup_k \rho_k}$ and $\cap_i \sigma_i \to \cup_j \tau_j \in L^{\leqslant}_{\cap_i \sigma_i \to \cup_j \tau_j}$. For any union of ANFs $\cup_k \rho_k$, let $P_1(\cup_k \rho_k)$ be the predicate *"for any σ, τ, such that $\sigma \leqslant \tau$ and $\tau \in L^{\leqslant}_{\cup_k \rho_k}$ we have that $\sigma \in L^{\leqslant}_{\cup_k \rho_k}$"*, and for any ANF $\cap_i s_i \to \cup_j \tau_j$, let $P_2(\cap_i s_i \to \cup_j \tau_j)$ be the predicate *"for any σ, τ, such that $\sigma \leqslant \tau$ and $\tau \in L^{\leqslant}_{\cap_i s_i \to \cup_j \tau_j}$ we have that $\sigma \in L^{\leqslant}_{\cap_i s_i \to \cup_j \tau_j}$"*. The proof is achieved in two steps.
  Step 1: we prove by induction on the subtyping rules of the (sub)type theory Ξ that:
  - If $P_1(\cup_j \tau_j)$ then $P_2(\cap_i s_i \to \cup_j \tau_j)$;
  - If, for all arrow types in $\cup_k \rho_k$ we have $P_2(\rho_k)$, then $P_1(\cup_k \rho_k)$.

  Step 2: using Step 1, we can prove by mutual induction on the definitions of $L^{\leqslant}_{\cap_i s_i \to \cup_j \tau_j}$ and $L^{\leqslant}_{\cup_k \tau_k}$ that P_1 and P_2 always hold.

The next lemma is useful for proving the completeness of $\mathcal{A}_1$.

Lemma 26. *If all the σ_i and τ_i are ANFs, then $\cap_i \sigma_i \leqslant \cup_j \tau_j$ iff $\exists i, j, \sigma_i \leqslant \tau_j$.*

Proof. If $\exists i, j, \sigma_i \leqslant \tau_j$, it is clear that $\cap_{1 \leqslant i \leqslant m} \sigma_i \leqslant \cup_{1 \leqslant j \leqslant n} \tau_j$. Reciprocally, we proceed by strong recurrence on $m + n$.

- *If $m = 1$ and $n = 1$, then it is clear that $\sigma_1 \leqslant \tau_1$;*
- *If $m = 1$ and $n > 1$, then, by Lemma 22, either $\cap_i \sigma_i \leqslant \cup_{1 \leqslant j \leqslant \lfloor \frac{n}{2} \rfloor} \tau_j$, or $\cap_i \sigma_i \leqslant \cup_{\lfloor \frac{n}{2} \rfloor + 1 \leqslant j \leqslant n} \tau_j$;*
- *If $m > 1$ and $n \geqslant 1$, then, by Lemma 25, either $\cap_{1 \leqslant i \leqslant \lfloor \frac{m}{2} \rfloor} \sigma_i \leqslant \tau_1$, or $\cap_{\lfloor \frac{m}{2} \rfloor + 1 \leqslant i \leqslant m} \sigma_i \leqslant \tau_1$.*

Lemma 27.

1. $\mathcal{R}_2 \circ \mathcal{R}_1 = \mathcal{R}_1 \circ \mathcal{R}_2$;
2. $\mathcal{R}_3 \circ \mathcal{R}_1 = \mathcal{R}_1 \circ \mathcal{R}_3$;
3. $\mathcal{R}_4 \circ \mathcal{R}_1 = \mathcal{R}_1 \circ \mathcal{R}_4 \circ \mathcal{R}_1$.

Proof.

- *(i) and (ii). We prove that the term rewriting system obtained by merging* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *(resp.* $\mathcal{R}_3$*) is strongly normalizing and that all the critical pairs are locally confluent. According to Newman's lemma, we get a confluent term rewriting system. Therefore,* $\mathcal{R}_2 \circ \mathcal{R}_1 = \mathcal{R}_1 \circ \mathcal{R}_2$ *(resp.* $\mathcal{R}_3 \circ \mathcal{R}_1 = \mathcal{R}_1 \circ \mathcal{R}_3$*);*
- *(iii). None of the rewriting rules in* $\mathcal{R}_4$ *create any new redex for* $\mathcal{R}_1$*, therefore we have that* $\mathcal{R}_4 \circ \mathcal{R}_1 = \mathcal{R}_1 \circ \mathcal{R}_4 \circ \mathcal{R}_1$*.*

Corollary 28.

1. $\mathcal{R}_2 \circ \mathcal{R}_4 \circ \mathcal{R}_1 = \mathcal{R}_1 \circ \mathcal{R}_2 \circ \mathcal{R}_4 \circ \mathcal{R}_1$*;*
2. $\mathcal{R}_3 \circ \mathcal{R}_4 \circ \mathcal{R}_1 = \mathcal{R}_1 \circ \mathcal{R}_3 \circ \mathcal{R}_4 \circ \mathcal{R}_1$*.*

Proof. Immediate from Lemma 27.

We can finally state completeness of the algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$.

Theorem 29 ($\mathcal{A}_1, \mathcal{A}_2$'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* $\tau \not\sim \omega$ *be ANFs. 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 since $\sigma' \leqslant \tau'$*,* $\cup_i(\cap_j \sigma_{i,j}) \leqslant \cup_j \cap_h (\cup_k \tau_{h,k})$*. The proof proceeds by mutual induction.*

1. *Algorithm* $\mathcal{A}_1$*. If any of the* $\tau_{h,k}$ *is equivalent to* ω*, then* $\cup_k \tau_{h,k}$ *is equivalent to* ω*. Then by Lemma 24,* $\cup_k \tau_{h,k}$ *has been rewritten into* ω*. As we suppose* $\tau_{h,k}$ *has not been "erased" by* $\mathcal{R}_1$*, we can deduce that* $\cap_h(\cup_k \tau_{h,k}) \equiv \omega$*. The algorithm accepts, so we can conclude.*
   On the other hand, we know by Corollary 28 that $\mathcal{R}_1(\sigma_{i,j}) \equiv \sigma_{i,j}$ *and* $\mathcal{R}_1(\tau_{h,k}) \equiv \tau_{h,k}$*, so if none of the* $\tau_{h,k}$ *are equivalent to* ω*, then we can safely call* $\mathcal{A}_2(\sigma_{i,j} \leqslant \tau_{h,k})$*. Moreover, we know by Lemmas 16 and 26, that for all* i *and* h*, there exists some* j *and some* k*, such that* $\sigma_{i,j} \leqslant \tau_{h,k}$*. As the induction hypothesis states that* $\mathcal{A}_2$ *decides subtyping for any of the possible* $\sigma_{i,j} \leqslant \tau_{h,k}$*, we can conclude;*
2. *Algorithm* $\mathcal{A}_2$*. We proceed by case analysis on the algorithm* $\mathcal{A}_2$*.*
   - *Case* $\omega \leqslant \tau$*: by hypothesis,* $\omega \not\leqslant \tau$*, so this case is impossible;*
   - *Case* $\phi \leqslant \phi'$*: according to Lemma 22, the only atomic variable* ϕ'' *such that* $\phi \leqslant \phi''$ *is* ϕ *itself;*
   - *Case* $\sigma \to \tau \leqslant \phi$*: impossible by inspecting* $L^{\leqslant}_{\phi}$*;*
   - *Case* $\phi \leqslant \sigma \to \tau$*: by inspecting* $L^{\geqslant}_{\phi}$*, we have that* $\phi \leqslant \sigma \to \tau$ *iff* $\sigma \to \tau \sim \omega$*. However, this is impossible because of the hypothesis* $\sigma \to \tau \not\sim \omega$*;*
   - *Case* $\sigma \to \tau \leqslant \sigma' \to \tau'$*: we know that* $\tau' \not\sim \omega$*, therefore, by Corollary 23, we have that* $\tau \leqslant \tau'$*, and* $\sigma' \leqslant \sigma$*. Since we know that* $\sigma, \sigma', \tau, \tau'$ *have already been rewritten by* $\mathcal{R}_1$*, and that they already are in CANF or DANF, it is no need to preprocess them through* $\mathcal{R}_2 \circ \mathcal{R}_4 \circ \mathcal{R}_1$*, or* $\mathcal{R}_3 \circ \mathcal{R}_4 \circ \mathcal{R}_1$*. By induction hypothesis,* $\mathcal{A}_1$ *decides that* $\sigma' \leqslant \sigma$ *and* $\tau \leqslant \tau'$*, so we can conclude.*