The Delta-framework

Furio Honsell, Luigi Liquori, Claude Stolze, Ivan Scagnetto

The Δ-framework

Furio Honsell
Dept. of Mathematics, Computer Science and Physics, University of Udine, Via delle Scienze, 206, 33100 Udine, Italy
furio.honsell@uniud.it

Luigi Liquori
Université Côte d'Azur, INRIA Sophia Antipolis - Méditerranée 2004 Route des Lucioles - BP 93 FR-06902 Sophia Antipolis, France
Luigi.Liquori@inria.fr

Claude Stolze
Université Côte d'Azur, INRIA Sophia Antipolis - Méditerranée 2004 Route des Lucioles - BP 93 FR-06902 Sophia Antipolis, France
Claude.Stolze@inria.fr

Ivan Scagnetto
Dept. of Mathematics, Computer Science and Physics, University of Udine, Via delle Scienze, 206, 33100 Udine, Italy
ivan.scagnetto@uniud.it

Abstract

We introduce the Δ-framework, LF_Δ, a dependent type theory based on the Edinburgh Logical Framework LF, extended with the *strong proof-functional connectives*, *i.e.* strong intersection, minimal relevant implication and strong union. Strong proof-functional connectives take into account the shape of logical proofs, thus reflecting polymorphic features of proofs in formulæ. This is in contrast to classical or intuitionistic connectives where the meaning of a compound formula depends only on the truth value or the provability of its subformulæ. Our framework encompasses a wide range of type disciplines. Moreover, since relevant implication permits to express subtyping, LF_Δ subsumes also Pfenning's refinement types. We discuss the design decisions which have led us to the formulation of LF_Δ, study its metatheory, and provide various examples of applications. Our strong proof-functional type theory can be plugged in existing common proof assistants.

Theory of computation → Logic → Logic and verification

Logic of programs, type theory, λ-calculus

1 Introduction

This paper provides a unifying framework for two hitherto unreconciled understandings of types: *i.e.* types-as-predicates *à la* Curry and types-as-propositions (sets) *à la* Church. The key to our unification consists in introducing *strong proof-functional connectives* [40, 3, 4] in a dependent type theory such as the Edinburgh Logical Framework (LF) [22]. Both Logical Frameworks and Proof-Functional Logics consider proofs as first class citizens, albeit differently. Strong proof-functional connectives take seriously into account the shape of logical proofs, thus allowing for polymorphic features of proofs to be made explicit in formulæ. Hence they provide a finer semantics than classical/intuitionistic connectives, where the meaning of a compound formula depends only on the *truth value* or the *provability* of its subformulæ. However, existing approaches to strong proof-functional connectives are all quite idiosyncratic in mentioning proofs. Existing Logical Frameworks, on the other hand,

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

provide a uniform approach to proof terms in object logics, but they do not fully capitalize on subtyping.

This situation calls for a natural combination of the two understandings of types, which should benefit both worlds. On the side of Logical Frameworks, the expressive power of the metalanguage would be enhanced thus allowing for shallower encodings of logics, a more principled use of subtypes [37], and new possibilities for formal reasoning in existing interactive theorem provers. On the side of type disciplines for programming languages, a principled framework for proofs would be provided, thus supporting a uniform approach to "proof reuse" practices based on type theory [38, 12, 20, 9, 6].

Therefore, in this paper, we extend LF with the connectives of *strong intersection*, *strong union*, and *minimal relevant implication* of Proof-Functional Logics [40, 3, 4]. We call this extension the Δ-framework (LF$_\Delta$), since it builds on the Δ-calculus [31]. Moreover, we illustrate by way of examples, that LF$_\Delta$ subsumes many expressive type disciplines in the literature [37, 3, 4, 38, 12].

It is not immediate to extend the judgments-as-type, Curry-Howard paradigm to logics supporting strong proof-functional connectives, since these connectives need to compare the shapes of derivations and do not just take into account the provability of propositions, *i.e.* the inhabitation of the corresponding type. In order to capture successfully strong logical connectives such as $\cap$ or $\cup$, we need to be able to express the rules:

$$\frac{\mathcal{D}_1 : A \quad \mathcal{D}_2 : B \quad \mathcal{D}_1 \equiv \mathcal{D}_2}{A \cap B}\ (\cap I) \qquad \frac{\mathcal{D}_1 : A \supset C \quad \mathcal{D}_2 : B \supset C \quad A \cup B \quad \mathcal{D}_1 \equiv \mathcal{D}_2}{C}\ (\cup E)$$

where $\equiv$ is a suitable equivalence between logical proofs. Notice that the above rules suggest immediately intriguing applications in polymorphic constructions, *i.e.* the same evidence can be used as a proof for different statements. Pottinger [40] was the first to study the strong connective $\cap$. He contrasted it to the intuitionistic connective $\wedge$ as follows: *"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 … (while) … to assert $A \wedge B$ is to assert that one has a pair of reasons, the first of which is a reason for asserting A and the second of which is a reason for asserting B"*. A logical theorem involving intuitionistic conjunction which does not hold for strong conjunction is $(A \supset A) \wedge (A \supset B \supset A)$, otherwise there should exist a closed λ-term having simultaneously both one and two abstractions. Lopez-Escobar [32] and Mints [35] investigated extensively logics featuring both strong and intuitionistic connectives especially in the context of *realizability* interpretations.

Dually, it is in the $\cup$-elimination rule that proof equality needs to be checked. Following Pottinger, we could say that *asserting $(A \cup B) \supset C$ is to assert that one has a reason for $(A \cup B) \supset C$, which is also a reason to assert $A \supset C$ and $B \supset C$.* The two connectives differ since the intuitionistic theorem $((A \supset B) \vee B) \supset A \supset B$ is not derivable for $\cup$, otherwise there would exist a term which behaves both as $\mathbf{I}$ and as $\mathbf{K}$.

Following Barbanera and Martini [4], *Minimal Relevant Implication*, $\supset_r$, can be viewed as a special case of implication whose related function space is the simplest possible one, namely the one containing only the identity function. The operators $\supset$ and $\supset_r$ differ, since $A \supset_r B \supset_r A$ is not derivable. Relevant implication allows for a natural introduction of subtyping, in that $A \supset_r B$ morally means $A \leqslant B$. Relevant implication amounts to a notion of "proof-reuse". Combining the remarks in [4, 3], minimal relevant implication, strong intersection and strong union correspond respectively to the implication, conjunction and disjunction operators of Meyer and Routley's Minimal Relevant Logic B^+ [34][1].

[1] A terminological comment is in order. We refer to $(\supset_r)$ as "relevant implication" in order to be faithful

$$\frac{B \vdash M : \sigma \quad B \vdash M : \tau}{B \vdash M : \sigma \cap \tau}\ (\cap I) \qquad \frac{B \vdash M : \sigma \cap \tau}{B \vdash M : \sigma}\ (\cap E_l) \qquad \frac{B \vdash M : \sigma \cap \tau}{B \vdash M : \tau}\ (\cap E_r)$$

$$\frac{B \vdash M : \sigma}{B \vdash M : \sigma \cup \tau}\ (\cup I_l) \qquad \frac{B \vdash M : \tau}{B \vdash M : \sigma \cup \tau}\ (\cup I_r)$$

$$\frac{B, x{:}\sigma \vdash M : \rho \quad B, x{:}\tau \vdash M : \rho \quad B \vdash N : \sigma \cup \tau}{B \vdash M[N/x] : \rho}\ (\cup E) \qquad \frac{B \vdash M : \sigma \quad \sigma \leq \tau}{B \vdash M : \tau}\ (Sub)$$

$$\frac{x{:}\sigma \in B}{B \vdash x : \sigma}\ (Var) \qquad \frac{B \vdash M : \sigma \to \tau \quad B \vdash N : \sigma}{B \vdash M\,N : \tau}\ (App) \qquad \frac{B, x{:}\sigma \vdash M : \tau}{B \vdash \lambda x.M : \sigma \to \tau}\ (Abs)$$

(1) $\sigma \leqslant \sigma \cap \sigma$

(2) $\sigma \cup \sigma \leqslant \sigma$

(3) $\sigma \cap \tau \leqslant \sigma, \sigma \cap \tau \leqslant \tau$

(4) $\sigma \leqslant \sigma \cup \tau, \tau \leqslant \sigma \cup \tau$

(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) $\sigma_2 \leqslant \sigma_1, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \to \tau_1 \leqslant \sigma_2 \to \tau_2$

Figure 1 The type assignment system $\mathcal{B}$ of [3] and the subtype theory Ξ

Strong connectives arise naturally in investigating the propositions-as-types analogy for intersection and union type assignment systems. Intersection types were introduced by Coppo, Dezani *et al.* in the late 70's [13, 15, 16, 5] to support a form of *ad hoc* polymorphism, for untyped λ-calculi, *à la* Curry. Intersection types were used originally as an (undecidable) type assignment system for pure λ-calculi, *i.e.* for finitary descriptions of denotational semantics [14]. This line of research was later explored by Abramsky [1] in a full-fledged Stone duality. Union types were introduced semantically, by MacQueen, Plotkin, and Sethi [33, 3]. In [3] strong intersection, union and subtyping were thoroughly studied in the context of type-assignment systems, see Figure 1. A classical example of the expressiveness of union types is due to Pierce [38]: without union types, the best information we can get for (Is_0 Test) is a boolean type.

$$\begin{array}{rcl} \text{Test} & \stackrel{def}{=} & \text{if } b \text{ then } 1 \text{ else } -1 : Pos \cup Neg \\ \text{Is\_0} & : & (Neg \to F) \cap (Zero \to T) \cap (Pos \to F) \\ (\text{Is\_0 Test}) & : & F \end{array}$$

Designing a λ-calculus *à la* Church with intersection and union types is problematic. The usual approach of simply adding types to binders does not work, as shown in Figure 2. Same difficulties can be found with union types. Intersection and union type disciplines

to the original logical literature, since this constructor satisfies the logical properties of implication in the minimal relevant logical system introduced in [34]. And precisely in this sense it was used later in [4]. This use of the word "relevant" is therefore considerably *stronger* than, but not totally unrelated to, the one arising in the context of λI$-$*calculus* and linear logic, where it expresses the requirement that the variable "is used at least once" in the function, in contrast to affine "at most one use" and linear "exactly one use".

started to be investigated in a explicitly typed programming language settings *à la* Church, much later by Reynolds and Pierce [41, 38], Wells *et al.* [48, 49], Liquori *et al.* [29, 18], Frisch *et al.* [21] and Dunfield [19]. From a logical point of view, there are many proposals to find a suitable logics to fit intersection: among them we cite [35, 37, 47, 42, 36, 11, 10, 39].

$$\dfrac{\dfrac{\dfrac{}{x{:}\sigma \vdash x{:}\sigma}\,(Var)}{\vdash \lambda x{:}\sigma.x{:}\sigma \to \sigma}\,(\to I) \qquad \dfrac{\dfrac{}{x{:}\tau \vdash x{:}\tau}\,(Var)}{\vdash \lambda x{:}\tau.x{:}\tau \to \tau}\,(\to I)}{\vdash \lambda x{:}???.x{:}(\sigma \to \sigma) \cap (\tau \to \tau)}\,(\cap I)$$

Figure 2 Polymorphic identity

The LF_Δ, introduced in this paper extends [31] with union types, dependent types and minimal relevant implication. The novelty of LF_Δ in the context of Logical Frameworks, lies in the full-fledged use of strong proof-functional connectives, which to our knowledge has never been explored before. Clearly, all Δ-terms have a computational counterpart.

Pfenning's work on Refinement Types [37] pioneered an extension of the Edinburgh Logical Framework with subtyping and intersection types. His approach capitalises on a tame and essentially *ad hoc* notion of subtyping, but the logical strength of that system does not go beyond the LF (*i.e.* simple types). The logical power of LF_Δ allows to type all strongly normalizing terms. Furthermore, subtyping in LF_Δ arises naturally as a derived notion from the more fundamental concept of minimal relevant implication, as illustrated in Section 2.

Miquel [36] discusses an extension of the Calculus of Constructions with implicit typing, which subsumes a kind of proof-functional intersection. His approach has opposite motivations to ours. While LF_Δ provides a Church-style version of Curry-style type assignment systems, Miquel's Implicit Calculus of Constructions encompasses some features of Curry-style systems in an otherwise Church-style Calculus of Constructions. In LF_Δ we can discuss also *ad hoc* polymorphism, while in the Implicit Calculus only structural polymorphism is encoded. Indeed, he cannot assign the type $((\sigma \cap \tau) \to \sigma) \cap (\rho \to \rho))$ to the identity $\lambda x.x$ [28]. Kopylov [27] adds a dependent intersection type constructor $x{:}A \cap B[x]$ to NuPRL, allowing the resulting system to support dependent records (which are a very useful data structure to encode mathematics). The implicit product-type of Miquel, together with the dependent intersection type of Kopylov, and a suitable equality-type is used by Stump [46] to enrich the impredicative second-order system $\lambda P2$, in order to derive induction.

In order to achieve our goals, we could have carried out simply the encoding of LF_Δ in LF. But, due to the side-conditions characterizing proof-functional connectives, this would have be achieved only through a deep encoding. As an example of this, in Figure 8, we give an encoding of a subsystem of [3], where subtyping has been simulated using relevant arrows. This encoding illustrates the expressive power of LF in treating proofs as first-class citizens, and it was also a source of inspiration for LF_Δ.

All the examples discussed in this paper have been checked by an experimental proof development environment for LF_Δ [45] (see Bull and Bull-Subtyping in [44]).

Synopsis. In Section 2, we introduce LF_Δ and outline its metatheory, together with a discussion of the main design decisions. In Section 3, we provide the motivating examples. In Section 4, we outline the details of the implementation and future work.

2 The Δ-framework: LF with proof-functional operators

The syntax of LF_Δ pseudo-terms is given in Figure 3. For the sake of simplicity, we suppose that α-convertible terms are equal. Signatures and contexts are defined as finite sequence of declarations, like in LF. Observe that we could formulate LF_Δ in the style of [23], using only

Kinds				Objects				
K	$::=$	$\mathsf{Type} \mid \Pi x{:}\sigma.K$	as in LF	Δ	$::=$	$c \mid x \mid \lambda x{:}\sigma.\Delta \mid \Delta\,\Delta \mid$	as in LF
Families						$\lambdabar x{:}\sigma.\Delta \mid$	relevant abstraction
σ, τ	$::=$	$a \mid \Pi x{:}\sigma.\tau \mid \sigma\,\Delta \mid$	as in LF			$\Delta\,{}^{\mathsf{r}}\Delta \mid$	relevant application
		$\sigma \to^{\mathsf{r}} \tau \mid$	relevant family			$\langle \Delta \ , \ \Delta \rangle \mid$	intersection objects
		$\sigma \cap \tau \mid$	intersection family			$[\Delta \ , \ \Delta] \mid$	union objects
		$\sigma \cup \tau$	union family			$pr_l\,\Delta \mid pr_r\,\Delta \mid$	projections objects
						$in_l^{\sigma}\,\Delta \mid in_r^{\sigma}\,\Delta$	injections objects

Figure 3 The syntax of the Δ-framework

$$
\begin{array}{lcl@{\qquad}lcl@{\qquad}lcl}
\wr \langle \Delta_1 \ , \ \Delta_2 \rangle \wr & \stackrel{def}{=} & \wr \Delta_1 \wr & \wr [\Delta_1 \ , \ \Delta_2] \wr & \stackrel{def}{=} & \wr \Delta_1 \wr & \wr pr_i\,\Delta \wr & \stackrel{def}{=} & \wr \Delta \wr \\
\wr \lambdabar x{:}\sigma.\Delta \wr & \stackrel{def}{=} & \lambda x.\wr \Delta \wr & \wr \Delta_1\,\Delta_2 \wr & \stackrel{def}{=} & \wr \Delta_1 \wr \wr \Delta_2 \wr & \wr in_i\,\Delta \wr & \stackrel{def}{=} & \wr \Delta \wr \\
\wr \lambda x{:}\sigma.\Delta \wr & \stackrel{def}{=} & \lambda x.\wr \Delta \wr & \wr \Delta_1\,{}^{\mathsf{r}}\,\Delta_2 \wr & \stackrel{def}{=} & \wr \Delta_2 \wr & \wr c \wr & \stackrel{def}{=} & c \\
 & & & & & & \wr x \wr & \stackrel{def}{=} & x
\end{array}
$$

Figure 4 The essence function

canonical forms and without reductions, but we prefer to use the standard LF format to support better intuition. There are three proof-functional objects, namely strong conjunction (typed with $\sigma \cap \tau$) with two corresponding projections, strong disjunction (typed with $\sigma \cup \tau$) with two corresponding injections, and strong (or relevant) λ-abstraction (typed with $\to^{\mathsf{r}}$). Indeed, a relevant implication is not a dependent one because the essence of the inhabitants of type $\sigma \to^{\mathsf{r}} \tau$ is essentially the identity function as enforced in the typing rules. Note that injections in_i need to be decorated with the injected type σ in order to ensure the unicity of typing.

We need to generalize the notion of *essence*, introduced in [17, 30] to syntactically connect pure λ-terms (denoted by M) and type annotated LF_Δ terms (denoted by Δ). The essence function compositionally erases all type annotations, see Figure 4.

One could argue that the choice of Δ_1 in the definition of strong pairs/co-pairs is arbitrary and could have been replaced with Δ_2: however, the typing rules will ensure that, if $\langle \Delta_1 \ , \ \Delta_2 \rangle$ (resp. $[\Delta_1 \ , \ \Delta_2]$) is typable, then we have that $\wr \Delta_1 \wr =_\eta \wr \Delta_2 \wr$. Thus, strong pairs/co-pairs are constrained. The rule for the essence of a relevant application is justified by the fact that the operator amounts to just a type decoration.

The six basic reductions for LF_Δ objects appear on the left in Figure 5. Congruence rules are as usual, except for the two cases dealing with pairs and co-pairs which appear on the right of Figure 5. Here redexes need to be reduced "in parallel" in order to preserve identity of essences in the components. We denote by $=_\Delta$ the symmetric, reflexive, and transitive closure of $\to_\Delta$, *i.e.* the compatible closure of the reduction induced by the first six rules on the left in Figure 5, with the addition of the last two congruence rules in the same figure. In order to make this definition truly functional as well as to be able to prove a simple subject reduction result, we need to constrain pairs and co-pairs, *i.e.* objects of the form

$$
\begin{array}{rcl}
(\lambda x{:}\sigma.\Delta_1)\,\Delta_2 & \longrightarrow_{\beta} & \Delta_1[\Delta_2/x] \\
pr_l\,\langle \Delta_1 \,,\, \Delta_2\rangle & \longrightarrow_{pr_l} & \Delta_1 \\
pr_r\,\langle \Delta_1 \,,\, \Delta_2\rangle & \longrightarrow_{pr_r} & \Delta_2 \\
[\Delta_1 \,,\, \Delta_2]\, in_l^{\sigma}\,\Delta_3 & \longrightarrow_{in_l} & \Delta_1\,\Delta_3 \\
[\Delta_1 \,,\, \Delta_2]\, in_r^{\sigma}\,\Delta_3 & \longrightarrow_{in_r} & \Delta_2\,\Delta_3 \\
(\lambda^{\mathsf{r}} x{:}\sigma.\Delta_1)\,{}^{\mathsf{r}}\Delta_2 & \longrightarrow_{\beta r} & \Delta_1[\Delta_2/x]
\end{array}
$$

$$
\frac{\Delta_1 \to_\Delta \Delta_1' \quad \Delta_2 \to_\Delta \Delta_2' \quad \wr\,\Delta_1'\,\wr \equiv \wr\,\Delta_2'\,\wr}{\langle \Delta_1 \,,\, \Delta_2\rangle \to_\Delta \langle \Delta_1' \,,\, \Delta_2'\rangle}\ (Congr_{\cap})
$$

$$
\frac{\Delta_1 \to_\Delta \Delta_1' \quad \Delta_2 \to_\Delta \Delta_2' \quad \wr\,\Delta_1'\,\wr \equiv \wr\,\Delta_2'\,\wr}{[\Delta_1 \,,\, \Delta_2] \to_\Delta [\Delta_1' \,,\, \Delta_2']}\ (Congr_{\cup})
$$

Figure 5 The reduction semantics

$\langle \Delta_i \,,\, \Delta_j\rangle$ and $[\Delta_i \,,\, \Delta_j]$ to have congruent components up-to erasure of type annotations. This is achieved by imposing $\wr\,\Delta_i\,\wr \equiv \wr\,\Delta_j\,\wr$ in both constructs. We will therefore assume that such pairs and co-pairs are simply not well defined terms, if the components have a different "infrastructure". The effects of this choice are reflected in the congruence rules in the reduction relation, in order to ensure that reductions can only be carried out in parallel along the two components.

The restriction on reductions in pairs/co-pairs and the new constructs do not cause any problems in showing that $\to_\Delta$ is locally confluent:

▶ **Theorem 1** (Local confluence).
The reduction relation on well-formed LF_Δ*-terms is locally confluent.*

The extended type theory LF_Δ is a formal system for deriving judgements of the forms:

$$
\begin{array}{rcll}
 & \vdash & \Sigma & \Sigma \text{ is a valid signature} \\
 & \vdash_\Sigma & \Gamma & \Gamma \text{ is a valid context in } \Sigma \\
\Gamma & \vdash_\Sigma & K & K \text{ is a kind in } \Gamma \text{ and } \Sigma \\
\Gamma & \vdash_\Sigma & \sigma : K & \sigma \text{ has kind K in } \Gamma \text{ and } \Sigma \\
\Gamma & \vdash_\Sigma & \Delta : \sigma & \Delta \text{ has type } \sigma \text{ in } \Gamma \text{ and } \Sigma
\end{array}
$$

The set of rules for object formation is defined in Figure 6, while the sets of rules for signatures, contexts, kinds and families are defined as usual in the Appendix: all typing rules are syntax-directed. Note that proof-functionality is enforced by the essence side-conditions in rules $(\to^{\mathsf{r}} I)$, $(\cap I)$, and $(\cup E)$. In the rule $(Conv)$ we rely on the external notion of equality $=_\Delta$. An option could have be to add an internal notion of equality directly in the type system ($\Gamma \vdash_\Sigma \sigma =_\Delta \tau$), and prove that the external and the internal definitions of equality are equivalent, as was proved for semi-full Pure Type Systems [43]. Yet another possibility could be to compare type essences $\wr\,\sigma\,\wr =_\Delta \wr\,\tau\,\wr$, for a suitable extension of essence to types and kinds. Unfortunately, this would lead to undecidability of type checking, in connection with relevant implication, as the following example shows. Consider two constants c_1 of type $\sigma \to^{\mathsf{r}} (\Pi y{:}\sigma.\sigma)$ and c_2 of type $(\Pi y{:}\sigma.\sigma) \to^{\mathsf{r}} \sigma$: the following Δ-term is typable with σ and its essence is Ω.

$$
\Delta_\Omega \stackrel{def}{=} (\lambda x{:}\sigma.c_1\,{}^{\mathsf{r}}\,x\,x)\,(c_2\,{}^{\mathsf{r}}\,(\lambda x{:}\sigma.c_1\,{}^{\mathsf{r}}\,x\,x)) \qquad\qquad \wr\,\Delta_\Omega\,\wr = \Omega
$$

Since the intended meaning of relevant implication is "essentially" the identity, introducing variables or constants whose type is a relevant implication, amounts to assuming axioms corresponding to type inclusions such as those that equate σ and $\sigma \to \sigma$. As a consequence, β-equality of essences becomes undecidable. Thus, we rule out such options in relating relevant implications in LF_Δ to subtypes in the type assignment system $\mathcal{B}$ of [3].

Valid Objects

$$\frac{\vdash_\Sigma \Gamma \quad c{:}\sigma \in \Sigma}{\Gamma \vdash_\Sigma c : \sigma}\ (Const) \qquad \frac{\vdash_\Sigma \Gamma \quad x{:}\sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma}\ (Var)$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \Delta : \tau}{\Gamma \vdash_\Sigma \lambda x{:}\sigma.\Delta : \Pi x{:}\sigma.\tau}\ (\Pi I) \qquad \frac{\Gamma \vdash_\Sigma \Delta_1 : \Pi x{:}\sigma.\tau \quad \Gamma \vdash_\Sigma \Delta_2 : \sigma}{\Gamma \vdash_\Sigma \Delta_1\, \Delta_2 : \tau[\Delta_2/x]}\ (\Pi E)$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \Delta : \tau \quad \wr \Delta \wr =_\eta x}{\Gamma \vdash_\Sigma \lambda^{\mathsf{r}} x{:}\sigma.\Delta : \sigma \to^{\mathsf{r}} \tau}\ (\to^{\mathsf{r}} I) \qquad \frac{\Gamma \vdash_\Sigma \Delta_1 : \sigma \to^{\mathsf{r}} \tau \quad \Gamma \vdash_\Sigma \Delta_2 : \sigma}{\Gamma \vdash_\Sigma \Delta_1 \,{}^{\mathsf{r}}\Delta_2 : \tau}\ (\Pi^{\mathsf{r}} E)$$

$$\frac{\Gamma \vdash_\Sigma \Delta_1 : \sigma \quad \Gamma \vdash_\Sigma \Delta_2 : \tau \quad \wr \Delta_1 \wr =_\eta \wr \Delta_2 \wr}{\Gamma \vdash_\Sigma \langle \Delta_1 \ , \ \Delta_2 \rangle : \sigma \cap \tau}\ (\cap I) \quad \frac{\Gamma \vdash_\Sigma \Delta : \sigma \cap \tau}{\Gamma \vdash_\Sigma pr_l\, \Delta : \sigma}\ (\cap E_l) \quad \frac{\Gamma \vdash_\Sigma \Delta : \sigma \cap \tau}{\Gamma \vdash_\Sigma pr_r\, \Delta : \tau}\ (\cap E_r)$$

$$\frac{\Gamma \vdash_\Sigma \Delta : \sigma \quad \Gamma \vdash_\Sigma \sigma \cup \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma in_l^\tau\, \Delta : \sigma \cup \tau}\ (\cup I_l) \qquad \frac{\Gamma \vdash_\Sigma \Delta : \tau \quad \Gamma \vdash_\Sigma \sigma \cup \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma in_r^\sigma\, \Delta : \sigma \cup \tau}\ (\cup I_r)$$

$$\frac{\begin{array}{ll}\Gamma \vdash_\Sigma \Delta_1 : \Pi y{:}\sigma.\rho[in_l^\tau\, y/x] & \wr \Delta_1 \wr =_\eta \wr \Delta_2 \wr \\ \Gamma \vdash_\Sigma \Delta_2 : \Pi y{:}\tau.\rho[in_r^\sigma\, y/x] & \Gamma, x{:}\sigma \cup \tau \vdash_\Sigma \rho : \mathsf{Type}\end{array}}{\Gamma \vdash_\Sigma [\Delta_1 \ , \ \Delta_2] : \Pi x{:}\sigma \cup \tau.\rho}\ (\cup E) \qquad \frac{\begin{array}{l}\Gamma \vdash_\Sigma \Delta : \sigma \\ \Gamma \vdash_\Sigma \tau : \mathsf{Type} \quad \sigma =_\Delta \tau\end{array}}{\Gamma \vdash_\Sigma \Delta : \tau}\ (Conv)$$

Figure 6 The type rules for valid objects

2.1 Relating LF_Δ to $\mathcal{B}$

We compare and contrast certain design decisions of LF_Δ to the type assignment system $\mathcal{B}$ of [3]. The proof of strong normalization for LF_Δ will rely, in fact, on a forgetful mapping from LF_Δ to $\mathcal{B}$. As pointed out in [3], the elimination rule for union types in $\mathcal{B}$ breaks subject reduction for one-step β-reduction, but this can be recovered using a suitable parallel β-reduction. The well-known counter-example for one-step reduction, due to Pierce is

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

where I is the identity. In the typing context $B \stackrel{def}{=} x{:}(\sigma_1 \to \sigma_1 \to \tau) \cap (\sigma_2 \to \sigma_2 \to \tau), y{:}\rho \to (\sigma_1 \cup \sigma_2), z{:}\rho$, the first and the last terms can be typed with τ, while the terms in the fork cannot. The reason is that the subject in the conclusion of the $(\cup E)$ rule uses a context which can have more than one hole, as in the present case[2]. In LF_Δ, the formulation of the $(\cup E)$ rule takes a different route which does not trigger the counterexample. Indeed, we have introduction and elimination constructs $in_l\,, in_r$ and $[\ ,\ ]$ which allow to reduce the term only if we know that the argument, stripped of the introduction construct, has one of the types of the disjunction. Pierce's critical term can be expressed and typed in LF_Δ with the

---

[2] The problem would not arise if $(\cup E)$ is replaced by the rule schema

$$\frac{B, x_1{:}\sigma, \ldots, x_n{:}\sigma \vdash M : \rho \quad B, x_1{:}\tau, \ldots, x_n{:}\tau \vdash M : \rho \quad B \vdash N_i : \sigma \cup \tau \quad N_i =_\beta N_j \quad i, j = 1 \ldots n}{B \vdash M[N_1/x_1 \ldots N_n/x_n] : \rho}\ (\cup E')$$

Removing the non-static clause on the N_i's would yield a more permissive type system than $\mathcal{B}$.

following judgment (the full derivation is in the Appendix):

$$\Gamma \vdash_\Sigma [\underbrace{(\lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1)}_{\Delta_1}\ ,\ \underbrace{(\lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2)}_{\Delta_2}]\,(\,\underbrace{(\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)}_{\Delta_3}\ y\, z) : \tau$$

where $\Gamma \stackrel{def}{=} x{:}(\Pi x_1{:}\sigma_1.\Pi x_2{:}\sigma_1.\tau) \cap (\Pi x_1{:}\sigma_2.\Pi x_2{:}\sigma_2.\tau)$, $y{:}\rho \to \sigma_1 \cup \sigma_2, z{:}\rho$, and $\Sigma \stackrel{def}{=} \tau{:}\mathsf{Type}$. Notice that there is only one redex, namely $\Delta_3\, y$, and the reduction of this redex leads to $[\Delta_1, \Delta_2]\,(y\, z)$, and no other intermediate (untypable) Δ-terms are possible.

The following result will be useful in the following section.

▶ **Theorem 2.** *The system $\mathcal{B}$ without ω gives types only to strongly normalizing terms.*

A proof is embedded in Theorem 4.8 of [3]. It can also be obtained using the general computability method presented in [25] Section 4, by interpreting intersection and union types precisely as intersections and unions in the lattice of computability sets.

2.2 LF_Δ metatheory

LF_Δ can play the role of a Logical Framework only if decidable. Due to the lack of space, we list here only the main results: the complete list appears in the Appendix. The first important step states that if a Δ-term is typable, then its type is unique up to $=_\Delta$.

▶ **Theorem 3** (Unicity of types and kinds).
1. *If $\Gamma \vdash_\Sigma \Delta : \sigma$ and $\Gamma \vdash_\Sigma \Delta : \tau$, then $\sigma =_\Delta \tau$.*
2. *If $\Gamma \vdash_\Sigma \sigma : K$ and $\Gamma \vdash_\Sigma \sigma : K'$, then $K =_\Delta K'$.*

Strong normalization is proved as in LF. First we encode LF_Δ-terms into terms of the type assignment system $\mathcal{B}$ such that redexes in the source language correspond to redexes in the target language and we use Theorem 2. Then, we introduce two forgetful mappings, namely $\|\cdot\|$ and $|\cdot|$, defined in Figure 11 of the Appendix, to erase dependencies in types and to drop proof-functional constructors in Δ-terms and we conclude. Special care is needed in dealing with redexes occurring in type-dependencies, because these need to be flattened at the level of terms.

▶ **Theorem 4** (Strong normalization).
1. LF_Δ *is strongly normalizing, i.e.,*
   a. *If $\Gamma \vdash_\Sigma K$, then K is strongly normalizing.*
   b. *If $\Gamma \vdash_\Sigma \sigma : K$, then σ is strongly normalizing.*
   c. *If $\Gamma \vdash_\Sigma \Delta : \sigma$, then Δ is strongly normalizing.*
2. *Every strongly normalizing pure λ-term can be annotated so as to be the essence of a Δ-term.*

Local confluence and strong normalization entail confluence, so we have

▶ **Theorem 5** (Confluence). LF_Δ *is confluent, i.e.:*
1. *If $K_1 \longrightarrow^*_\Delta K_2$ and $K_1 \longrightarrow^*_\Delta K_3$, then $\exists K_4$ such that $K_2 \longrightarrow^*_\Delta K_4$ and $K_3 \longrightarrow^*_\Delta K_4$.*
2. *If $\sigma_1 \longrightarrow^*_\Delta \sigma_2$ and $\sigma_1 \longrightarrow^*_\Delta \sigma_3$, then $\exists \sigma_4$ such that $\sigma_2 \longrightarrow^*_\Delta \sigma_4$ and $\sigma_3 \longrightarrow^*_\Delta \sigma_4$.*
3. *If $\Delta_1 \longrightarrow^*_\Delta \Delta_2$ and $\Delta_1 \longrightarrow^*_\Delta \Delta_3$, then $\exists \Delta_4$ such that $\Delta_2 \longrightarrow^*_\Delta \Delta_4$ and $\Delta_3 \longrightarrow^*_\Delta \Delta_4$.*

Then, we have subject reduction, whose proof relies on technical lemmas about inversion and subderivation properties (see Appendix).

▶ **Theorem 6** (Subject reduction of LF_Δ).

1. *If* $\Gamma \vdash_\Sigma K$ *and* $K \to_\Delta K'$, *then* $\Gamma \vdash_\Sigma K'$.
2. *If* $\Gamma \vdash_\Sigma \sigma : K$ *and* $\sigma \to_\Delta \sigma'$, *then* $\Gamma \vdash_\Sigma \sigma' : K$.
3. *If* $\Gamma \vdash_\Sigma \Delta : \sigma$ *and* $\Delta \to_\Delta \Delta'$, *then* $\Gamma \vdash_\Sigma \Delta' : \sigma$.

Finally, we define a possible algorithm for checking judgements in LF_Δ by computing a type or a kind for a term, and then testing for definitional equality, *i.e.* $=_\Delta$, against the given type or kind. This is achieved by reducing both to their unique normal forms and checking that they are identical up to α-conversion. Therefore we finally have:

▶ **Theorem 7** (Decidability). *All the type judgments of* LF_Δ *are recursively decidable.*

Minimal Relevant Implications and Type Inclusion. Type inclusion and the rules of subtyping are related to the notion of minimal relevant implication, see [4, 17]. The insight is quite subtle, but ultimately very simple. This is what makes it appealing. The apparently intricate rules of subtyping and type inclusion, which occur in many systems, and might even appear *ad hoc* at times, can all be explained away in our principled approach, by proving that the relevant implication type is inhabited by a term whose essence is essentially a variable.

In the following theorem we show how relevant implication subsumes the type-inclusion rules of the theory Ξ of [3], without rules (5) and (13) (dealing with ω) and rule (10) (distributing $\cap$ over $\cup$) in Figure 1: we call Ξ' such restricted subtype theory. Note that the reason to drop subtype rule (10) is due to the fact that we cannot inhabit the type $\sigma \cap (\tau \cup \rho) \to^{\mathsf{r}} (\sigma \cap \tau) \cup (\sigma \cap \rho)$[3].

▶ **Theorem 8** (Type Inclusion). *The judgement* $\langle\rangle \vdash_\Sigma \Delta : \sigma \to^{\mathsf{r}} \tau$ *(where both* σ *and* τ *do not contain dependencies or relevant families) holds iff* $\sigma \leq \tau$ *holds in the subtype theory* Ξ' *of* $\mathcal{B}$ *enriched with new axioms of the form* $\sigma_1 \leq \sigma_2$ *for each constant* $c : \sigma_1 \to^{\mathsf{r}} \sigma_2 \in \Sigma$.

As far as the $\lambda^{\Pi\&}$ system of Refinement Types introduced by Pfenning in [37], we have the following theorem:

▶ **Corollary 9** (Pfenning's Refinement Types). *The judgment* $\vdash_\Sigma \sigma \leq \tau$ *in* $\lambda^{\Pi\&}$ *can be encoded in* LF_Δ *by adding a constant of type* $\sigma \to^{\mathsf{r}} \tau$ *to* Σ', *where the latter is the signature obtained from* Σ *by replacing each clause of the form* $a_1 :: a_2$ *or* $a_1 \leq a_2$ *in* Σ *by a constant of type* $a_1 \to^{\mathsf{r}} a_2$.

Moreover, while Pfenning needs to add explicitly the rules of subtyping (*i.e.* the theory of $\leq$) in $\lambda^{\Pi\&}$, we inherit them naturally in LF_Δ from the rules for minimal relevant implication.

3 Examples

As we have argued in the previous sections, the point of this paper is a uniform and principled approach to the encoding of a plethora of type disciplines and systems which ultimately stem or can capitalize from strong proof-functional connectives and subtyping. The framework LF_Δ, presented in this paper, is the first to accommodate all the examples

[3] To encompass also the subtype rule (10) of the type theory Ξ, besides adding a special constant, we can strengthen the form of the $(\cup E)$ type rule as follows:

$$\frac{\begin{array}{ll}\Gamma \vdash_\Sigma \Delta_1 : \Pi y{:}\chi \cap \sigma.\rho\,\langle pr_l\, y \;,\; in_l^\tau\;\, pr_r\, y\rangle & \wr \Delta_1 \wr =_\eta \wr \Delta_2 \wr \\ \Gamma \vdash_\Sigma \Delta_2 : \Pi y{:}\chi \cap \tau.\rho\,\langle pr_l\, y \;,\; in_r^\sigma\;\, pr_r\, y\rangle & \Gamma \vdash_\Sigma \rho : \Pi y{:}\chi \cap (\sigma \cup \tau).\mathsf{Type}\end{array}}{\Gamma \vdash_\Sigma [\Delta_1 \;,\; \Delta_2] : \Pi x{:}\chi \cap (\sigma \cup \tau).\rho\, x}\;(\cup E)$$

Similarly we can treat the remaining rules of the type theory $\mathit{\Pi}$ in [3].

Atomic propositions, non-atomic goals and non-atomic programs: α, γ_0, π_0 : Type
Goals and programs: $\gamma = \alpha \cup \gamma_0 \quad \pi = \alpha \cup \pi_0$
Constructors (implication, conjunction, disjunction).

$$\begin{array}{lcl}
\mathsf{impl} & : & (\pi \to \gamma \to \gamma_0) \cap (\gamma \to \pi \to \pi_0) \\
\mathsf{impl}_1 & = & \lambda x{:}\pi.\lambda y{:}\gamma.in_r^\alpha\,(pr_l\,\mathsf{impl}\ x\ y) \qquad \mathsf{impl}_2 = \lambda x{:}\gamma.\lambda y{:}\pi.in_r^\alpha\,(pr_r\,\mathsf{impl}\ x\ y) \\
\mathsf{and} & : & (\gamma \to \gamma \to \gamma_0) \cap (\pi \to \pi \to \pi_0) \\
\mathsf{and}_1 & = & \lambda x{:}\gamma.\lambda y{:}\gamma.in_r^\alpha\,(pr_l\,\mathsf{and}\ x\ y) \qquad \mathsf{and}_2 = \lambda x{:}\pi.\lambda y{:}\pi.in_r^\alpha\,(pr_r\,\mathsf{and}\ x\ y) \\
\mathsf{or} & : & (\gamma \to \gamma \to \gamma_0) \qquad \mathsf{or}_1 = \lambda x{:}\gamma.\lambda y{:}\gamma.in_r^\alpha\,(\mathsf{or}\ x\ y)
\end{array}$$

$\mathsf{solve}\ p\ g$ indicates that the judgment $p \vdash g$ is valid.
$\mathsf{bchain}\ p\ a\ g$ indicates that, if $p \vdash g$ is valid, then $p \vdash a$ is valid.

$$\mathsf{solve} \ : \ \pi \to \gamma \to \mathsf{Type} \qquad \mathsf{bchain} : \pi \to \alpha \to \gamma \to \mathsf{Type}$$

Rules for solve:

$$\begin{array}{lcl}
- & : & \Pi_{(p:\pi)(g_1,g_2:\gamma)}\mathsf{solve}\ p\ g_1 \to \mathsf{solve}\ p\ g_2 \to \mathsf{solve}\ p\ (\mathsf{and}_1\ g_1\ g_2) \\
- & : & \Pi_{(p:\pi)(g_1,g_2:\gamma)}\mathsf{solve}\ p\ g_1 \to \mathsf{solve}\ p\ (\mathsf{or}_1\ g_1\ g_2) \\
- & : & \Pi_{(p:\pi)(g_1,g_2:\gamma)}\mathsf{solve}\ p\ g_2 \to \mathsf{solve}\ p\ (\mathsf{or}_1\ g_1\ g_2) \\
- & : & \Pi_{(p_1,p_2:\pi)(g:\gamma)}\mathsf{solve}\ (\mathsf{and}_2\ p_1\ p_2)\ g \to \mathsf{solve}\ p_1\ (\mathsf{impl}_1\ p_2\ g) \\
- & : & \Pi_{(p:\pi)(a:\alpha)(g:\gamma)}\mathsf{bchain}\ p\ a\ g \to \mathsf{solve}\ p\ g \to \mathsf{solve}\ p\ (in_l^{\gamma_0}\ a)
\end{array}$$

Rules for bchain:

$$\begin{array}{lcl}
- & : & \Pi_{(a:\alpha)(g:\gamma)}\mathsf{bchain}\ (\mathsf{impl}_2\ g\ (in_l\,\pi_0 a))\ a\ g \\
- & : & \Pi_{(p_1,p_2:\pi)(a:\alpha)(g:\gamma)}\mathsf{bchain}\ p_1\ a\ g \to \mathsf{bchain}\ (\mathsf{and}_2\ p_1\ p_2)\ a\ g \\
- & : & \Pi_{(p_1,p_2:\pi)(a:\alpha)(g:\gamma)}\mathsf{bchain}\ p_2\ a\ g \to \mathsf{bchain}\ (\mathsf{and}_2\ p_1\ p_2)\ a\ g \\
- & : & \Pi_{(p:\pi)(a:\alpha)(g,g_1,g_2:\gamma)}\mathsf{bchain}\ (\mathsf{impl}_2\ (\mathsf{and}_1\ g_1\ g_2)\ p)\ a\ g \to \mathsf{bchain}\ (\mathsf{impl}_2\ g_1\ (\mathsf{impl}_2\ g_2\ p))\ a\ g \\
- & : & \Pi_{(p_1,p_2:\pi)(a:\alpha)(g,g_1:\gamma)}\mathsf{bchain}\ (\mathsf{impl}_2\ g_1\ p_1)\ a\ g \to \mathsf{bchain}\ (\mathsf{impl}_2\ g_1\ (\mathsf{and}_2\ p_1\ p_2))\ a\ g \\
- & : & \Pi_{(p_1,p_2:\pi)(a:\alpha)(g,g_1:\gamma)},\mathsf{bchain}\ (\mathsf{impl}_2\ g_1\ p_2)\ a\ g \to \mathsf{bchain}\ (\mathsf{impl}_2\ g_1\ (\mathsf{and}_2\ p_1\ p_2))\ a\ g
\end{array}$$

Figure 7 The LF_Δ encoding of Hereditary Harrop Formulæ

and counterexamples that have appeared in the literature. The complete developments of both the implementation of the Δ-framework and example encodings can be found in [44].

We start the section showing the expressive power of LF_Δ in encoding classical features of typing disciplines with strong intersection and union.

Auto application. The judgement $\vdash_{\mathcal{B}} \lambda x.x\,x : \sigma \cap (\sigma \to \tau) \to \tau$ in $\mathcal{B}$, is rendered in LF_Δ by the LF_Δ-judgement $\vdash_\Sigma \lambda x{:}\sigma \cap (\sigma \to \tau).(pr_r\,x)\,(pr_l\,x) : \sigma \cap (\sigma \to \tau) \to \tau$.

Polymorphic identity. The judgement $\vdash_{\mathcal{B}} \lambda x.x : (\sigma \to \sigma) \cap (\tau \to \tau)$ in $\mathcal{B}$, is rendered in LF_Δ by the judgement $\vdash_{\langle\rangle} \langle \lambda x{:}\sigma.x\ ,\ \lambda x{:}\tau.x \rangle : (\sigma \to \sigma) \cap (\tau \to \tau)$.

Commutativity of union. The judgement $\lambda x.x : (\sigma \cup \tau) \to (\tau \cup \sigma)$ in $\mathcal{B}$ is rendered in LF_Δ by the judgement $\lambda x{:}\sigma{\cup}\tau.[\lambda y{:}\sigma.in_r^\tau\,y\ ,\ \lambda y{:}\tau.in_l^\sigma\ y]\,x : (\sigma \cup \tau) \to (\tau \cup \sigma)$.

Pierce's expression of page 2. The expressive power of union types highlighted by Pierce is rendered in LF_Δ by

$$\begin{array}{lcl}
Neg : \mathsf{Type} \quad Zero : \mathsf{Type} \quad Pos : \mathsf{Type} \quad T : \mathsf{Type} \quad F : \mathsf{Type} \quad \mathrm{Test} : Pos \cup Neg \\
\mathrm{Is_0} & : & (Neg \to F) \cap (Zero \to T) \cap (Pos \to F) \\
\mathrm{Is_0_Test} & \stackrel{def}{=} & [\lambda x{:}Neg.(pr_l\,pr_l\,\mathrm{Is_0})\,x\ ,\ \lambda x{:}Pos.(pr_r\,\mathrm{Is_0})\,x]\,\mathrm{Test}
\end{array}$$

The above example illustrates the advantages of taking LF_Δ as a framework. In LF we would render it only encoding $\mathcal{B}$ deeply, ending up with the verbose code in pierce_program.v [44].

Hereditary Harrop Formulæ. The encoding of Hereditary Harrop's Formulæ is one of the motivating examples given by Pfenning for introducing refinement types in [37]. In LF_Δ it can be expressed as in Figure 7 and type checked in the environment [45] using our concrete syntax (file pfenning_harrop.bull [44]), without any reference to intersection types, by a subtle use of union types. We add also rules for solving and backchaining. Hereditary Harrop formulæ can be recursively defined using two mutually recursive syntactical objects called programs (π) and goals (γ):

$$\gamma \quad ::= \quad \alpha \mid \gamma \wedge \gamma \mid \pi \Rightarrow \gamma \mid \gamma \vee \gamma \qquad\qquad \pi \quad ::= \quad \alpha \mid \pi \wedge \pi \mid \gamma \Rightarrow \pi$$

Using Corollary 9, we can provide an alternative encoding of atoms, goals and programs which is more faithful to the one by Pfenning. Namely, we can introduce in the signature the constants $c_1 : \alpha \to^{\mathsf{r}} \gamma$ and $c_2 : \alpha \to^{\mathsf{r}} \pi$ in order to represent the axioms $atom \leq goal$ and $atom \leq prog$ in Pfenning's encoding. Our approach based on union types, while retaining the same expressivity permits to shortcut certain inclusions and to rule out also certain exotic goals and exotic programs. Indeed, for the purpose of establishing the adequacy of the encoding, it is sufficient to avoid variables involving union types in the derivation contexts.

Natural Deductions in Normal Form. The second motivating example for intersection types given in [37] is *natural deductions in normal form*. We recall that a natural deduction is in normal form if there are no applications of elimination rules of a logical connective immediately following their corresponding introduction, in the main branch of a subderivation. The encoding we give in LF_Δ is a slightly improved version of the one in [37]: as Pfenning, we restrict to the purely implicational fragment. As in the previous example, we use union types to define normal forms ($\mathrm{Nf}(A)$) either as pure elimination-deductions from hypotheses ($Elim(A)$) or normal form-deductions ($\mathrm{Nf}^0(A)$). As above we could have used also intersection types. This example is interesting in itself, being the prototype of the encoding of type systems using canonical and atomic syntactic categories [23] and also of Fitch Set Theory [26].

$$\begin{array}{rcl}
o & : & Type \quad \supset : o \to o \to o \quad Elim, \mathrm{Nf}^0 : o \to \mathsf{Type} \\
\mathrm{Nf} & \equiv & \Pi A{:}o.\mathrm{Nf}^0(A) \cup Elim(A) \\
\supset_I & : & \Pi A, B{:}o.(Elim(A) \to \mathrm{Nf}(B)) \to \mathrm{Nf}^0(A \supset B) \\
\supset_E & : & \Pi A, B{:}o.Elim(A \supset B) \to \mathrm{Nf}^0(A) \to Elim(B).
\end{array}$$

Adequacy, Canonical Forms, Exotic terms. In the presence of union types, we have to pay special attention to the exact formulation of Adequacy Theorems, as in the Harrop's formulæ example above. Otherwise exotic terms arise, such as $[\lambda x{:}\sigma.C(x)\ ,\ \lambda x{:}\tau.D(x)]\, y$, where $C(\cdot)$ and $D(\cdot)$ are distinct contexts (*i.e.* terms with holes), which cannot be naturally simplified even if $\wr C \wr \equiv \wr D \wr$. More work needs to be done to streamline how to exclude, or even capitalize on exotic terms.

Metacircular Encodings. The following diagram summarizes the network of adequate encodings/inclusions between LF_Δ, LF, and $\mathcal{B}$ that can be defined. We denote by $\mathcal{S}_1 \Longrightarrow \mathcal{S}_2$ the encoding of system $\mathcal{S}_1$ in system $\mathcal{S}_2$, where the label *sh* (resp. *dp*), denotes a shallow (resp. deep) embedding. The notation $\mathcal{S}_1 \hookrightarrow \mathcal{S}_2$ denotes that $\mathcal{S}_2$ is an extension of $\mathcal{S}_1$. Due to lack of space, but with the intention of providing a better formal understanding of the semantics of strong intersection and union types in a logical framework, we provide in Figure 8 a deep LF encoding of a presentation of $\mathcal{B}$ *à la* Church [17]. A shallow encoding of $\mathcal{B}$ in LF_Δ (file intersection_union.bull [44]) can be mechanically type checked in the environment [45]. A shallow encoding of LF in LF_Δ (file lf.bull) making essential use of intersection types can be also type checked.

$$\begin{tikzcd}
\mathrm{LF} \arrow[r, Rightarrow, "sh"] & \mathrm{LF}_\Delta \arrow[r, Rightarrow, "dp"] & \mathrm{LF} \\
\mathcal{B} \arrow[ur, Rightarrow, "sh"] \arrow[r, Rightarrow, "dp"] & \mathrm{LF} \arrow[u, hook] &
\end{tikzcd}$$

LF encoding of $\mathcal{B}$. Figure 8 presents a pure LF encoding of a presentation of $\mathcal{B}$ *à la* Church in Coq syntax using HOAS. We use HOAS in order to take advantage of the higher-order

```
(* Define our types *)
Axiom o : Set.
(* Axiom omegatype : o. *)
Axioms (arrow inter union : o → o → o).

(* Transform our types into LF types *)
Axiom OK : o → Set.

(* Define the essence equality as an equivalence relation *)
Axiom Eq : forall (s t : o), OK s → OK t → Prop.
Axiom Eqrefl : forall (s : o) (M : OK s), Eq s s M M.
Axiom Eqsymm : forall (s t : o) (M : OK s) (N : OK t), Eq s t M N → Eq t s N M.
Axiom Eqtrans : forall (s t u : o) (M : OK s) (N : OK t) (O : OK u), Eq s t M N → Eq t u N O → Eq s u M O.

(* constructors for arrow (→ I and → E) *)
Axiom Abst : forall (s t : o), ((OK s) → (OK t)) → OK (arrow s t).
Axiom App : forall (s t : o), OK (arrow s t) → OK s → OK t.

(* constructors for intersection *)
Axiom Proj_l : forall (s t : o), OK (inter s t) → OK s.
Axiom Proj_r : forall (s t : o), OK (inter s t) → OK t.
Axiom Pair : forall (s t : o) (M : OK s) (N : OK t), Eq s t M N → OK (inter s t).

(* constructors for union *)
Axiom Inj_l : forall (s t : o), OK s → OK (union s t).
Axiom Inj_r : forall (s t : o), OK t → OK (union s t).
Axiom Copair : forall (s t u : o) (X : OK (arrow s u)) (Y : OK (arrow t u)), OK (union s t) →
 Eq (arrow s u) (arrow t u) X Y → OK u.

(* define equality wrt arrow constructors *)
Axiom Eqabst : forall (s t s' t' : o) (M : OK s → OK t) (N : OK s' → OK t'),
 (forall (x : OK s) (y : OK s'), Eq s s' x y → Eq t t' (M x) (N y)) →
 Eq (arrow s t) (arrow s' t') (Abst s t M) (Abst s' t' N).
Axiom Eqapp : forall (s t s' t' : o) (M : OK (arrow s t)) (N : OK s) (M' : OK (arrow s' t')) (N' : OK s'),
 Eq (arrow s t) (arrow s' t') M M' → Eq s s' N N' → Eq t t' (App s t M N) (App s' t' M' N').

(* define equality wrt intersection constructors *)
Axiom Eqpair : forall (s t : o) (M : OK s) (N : OK t) (pf : Eq s t M N), Eq (inter s t) s (Pair s t M N pf) M.
Axiom Eqproj_l : forall (s t : o) (M : OK (inter s t)), Eq (inter s t) s M (Proj_l s t M).
Axiom Eqproj_r : forall (s t : o) (M : OK (inter s t)), Eq (inter s t) t M (Proj_r s t M).

(* define equality wrt union *)
Axiom Eqinj_l : forall (s t : o) (M : OK s), Eq (union s t) s (Inj_l s t M) M.
Axiom Eqinj_r : forall (s t : o) (M : OK t), Eq (union s t) t (Inj_r s t M) M.
Axiom Eqcopair : forall (s t u : o) (M : OK (arrow s u)) (N : OK (arrow t u)) (O : OK (union s t))
 (pf: Eq (arrow s u) (arrow t u) M N) (x : OK s),
 Eq s (union s t) x O → Eq u u (App s u M x) (Copair s t u M N O pf).
```

Figure 8 The LF encoding of $\mathcal{B}$ (Coq syntax)

features of the frameworks: other abstract syntax representation techniques would not be much different, but more verbose. The `Eq` predicate plays the same role of the essence function in LF_Δ, namely, it encodes the judgement that two proofs (*i.e.* two terms of type `(OK _)`) have the same structure. This is crucial in the `Pair` axiom (*i.e.* the introduction rule of the intersection type constructor) where we can inhabit the type `(inter s t)` only when the proofs of its component types `s` and `t` share the same structure (*i.e.* we have a witness of type `(Eq s t M N)`, where `M` has type `(OK s)` and `N` has type `(OK t)`). A similar role is played by the `Eq` premise in the `Copair` axiom (*i.e.* the elimination rule of the union type constructor). We have an `Eq` axiom for each proof rule. Examples of this encoding can be found in intersection_union.v [44].

4 Implementation and Future Work

In a previous paper [45], we have implemented in OCaml suitable algorithms for type reconstruction, as well as type checking. In [30] we have implemented the subtyping algorithm which extends the well-known Hindley algorithm for intersection types [24] with union types. The subtyping algorithm has been mechanically proved correct in Coq, extending the Bessai's mechanized proof of a subtyping algorithm for intersection types [8].

A Read-Eval-Print-Loop allows to define axioms and definitions, and performs some basic terminal-style features like error pretty-printing, subexpressions highlighting, and file loading. Moreover, it can type-check a proof or normalize it, using a strong reduction evaluator. We use the syntax of Pure Type Systems [7] to improve the compactness and the modularity of the kernel. Binders are implemented using de Brujin indexes. We implemented the conversion rule in the simplest way possible: when we need to compare types, we syntactically compare their normal form. Abstract and concrete syntax are mostly aligned: the concrete syntax is similar to the concrete syntax of Coq (see Bull and Bull-Subtyping [44]).

We are currently designing a higher-order unification algorithm for Δ-terms and a bidirectional refinement algorithm, similar to the one found in [2]. The refinement can be split into two parts: the essence refinement and the typing refinement. In the same way, there will be a unification algorithm for the essence terms, and a unification algorithm for Δ-terms. The bidirectional refinement algorithm aims to have partial type inference, and to give as much information as possible to a hypothetical solver, or the unifier. For instance, if we want to find a $?y$ such that $\vdash_\Sigma \langle \lambda x{:}\sigma.x \ , \ \lambda x{:}\tau.?y \rangle : (\sigma \to \sigma) \cap (\tau \to \tau)$, we can infer that $x{:}\tau \vdash ?y : \tau$ and that $\wr\, ?y \,\wr = x$.

LF_Δ in Canonical Form. We presented LF_Δ in the standard LF format in order to support intuition. It would be worthwhile however, to attempt to formulate LF_Δ in the style of [23], using only canonical forms without reductions, especially in view of Adequacy Theorems. The term constructs peculiar to LF_Δ would then introduce new clauses in the definition of canonical and atomic terms. The principle to follow in this task is that atomic terms synthesize their type, while canonical terms are checked against their type. We are currently exploring with the following extension:

$$\begin{array}{rcl} M & ::= & \ldots \mid \lambda^{\mathsf{r}} x.M \mid \langle M \ , \ M \rangle \mid [M \ , \ M] \mid in_l\, M \mid in_r\, M \\ R & ::= & \ldots \mid pr_l\, R \mid pr_r\, R \mid R \,{}^{\mathsf{r}}\, M \end{array}$$

Notice the somewhat surprising treatment of the $[\ ,\ ]$ constructor, which is not really an elimination construct but rather behaves as another form of abstraction. Accordingly hereditary substitution needs to be extended.

An intriguing issue raised by one of the referees is to explore the connections between strong implication and the *singleton type* of the identity function. This could lead also to an internalization of the essence function.

References

1 Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51(1):1–77, 1991.

2 Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. A bidirectional refinement algorithm for the calculus of (co)inductive constructions. *Logical Methods in Computer Science*, 8(1), 2012.

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

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

5 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.

6 Gilles Barthe and Olivier Pons. Type isomorphisms and proof reuse in dependent type theory. In *Foundations of Software Science and Computation Structures, 4th International Conference, FOSSACS 2001*, pages 57–71, 2001.

7 Stefano Berardi. *Towards a mathematical analysis of the Coquand–Huet calculus of constructions and the other systems in Barendregt's cube.* PhD thesis, Dipartimento Matematica, Universita di Torino, 1988.

8 Jan Bessai. Extracting a formally verified Subtyping Algorithm for Intersection Types from Ideals and Filters. Talk at COST Types, 2016.

9 Olivier Boite. Proof reuse with extended inductive types. In *Theorem Proving in Higher Order Logics, 17th International Conference, TPHOLs 2004*, pages 50–65, 2004.

10 Viviana Bono, Betti Venneri, and Lorenzo Bettini. A typed lambda calculus with intersection types. *Theor. Comput. Sci.*, 398(1-3):95–113, 2008.

11 Beatrice Capitani, Michele Loreti, and Betti Venneri. Hyperformulae, Parallel Deductions and Intersection Types. *BOTH, Electr. Notes Theor. Comput. Sci.*, 50(2):180–198, 2001.

12 Joshua E. Caplan and Mehdi T. Harandi. A logical framework for software proof reuse. In *SSR*, pages 106–113, 1995.

13 Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Arch. Math. Log.*, 19(1):139–156, 1978.

14 Mario Coppo, Mariangiola Dezani-Ciancaglini, Honsell Furio, and Longo Giuseppe. Extended Type Structures and Filter Lambda Models. In *Logic Colloquium*, pages 241–262, 1983.

15 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Patrick Sallé. Functional characterization of some semantic equalities inside λ-calculus. In *International Colloquium on Automata, Languages, and Programming*, pages 133–146. Springer-Verlag, 1979.

16 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Zietschrift für Mathematische Logik und Grundlagen der Mathematik*, 27(2-6):45–58, 1981.

17 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-Verlag, 2016.

18 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-Verlag, 2010.

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

20 Amy P. Felty and Douglas J. Howe. Generalization and reuse of tactic proofs. In *Proc. of Logic Programming and Automated Reasoning, 5th International Conference, LPAR*, pages 1–15, 1994.

21 Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping: Dealing set-theoretically with function, union, intersection, and negation types. *Journal of the ACM (JACM)*, 55(4):19, 2008.

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

23 Robert Harper and Daniel R. Licata. Mechanizing metatheory in a logical framework. *J. Funct. Program.*, 17(4-5):613–673, 2007.

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

25 Furio Honsell and Marina Lenisa. Semantical analysis of perpetual strategies in lambda-calculus. *Theor. Comput. Sci.*, 212(1-2):183–209, 1999.

26 Furio Honsell, Marina Lenisa, Luigi Liquori, and Ivan Scagnetto. Implementing Cantor's paradise. In *Proc. of Programming Languages and Systems - 14th Asian Symposium, APLAS*, pages 229–250, 2016.

27 Alexei Kopylov. Dependent intersection: a new way of defining records in type theory. In *Proc. of 18th Annual IEEE Symposium of Logic in Computer Science, LICS*, pages 86–95, 2003.

28 Luigi Liquori, Andreas Nuyts, and Claude Stolze. Privates communications, 2017.

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

30 Luigi Liquori and Claude Stolze. A decidable subtyping logic for intersection and union types. In *Proc of TTCS*, volume 10608 of *Lecture Notes in Computer Science*, pages 74–90. Springer-Verlag, 2017.

31 Luigi Liquori and Claude Stolze. The Delta-calculus: syntax and types. Research report, Inria, July 2018. URL: `https://arxiv.org/abs/1803.09660`.

32 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.

33 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.

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

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

36 Alexandre Miquel. The implicit calculus of constructions. In *TLCA*, pages 344–359, 2001.

37 Frank Pfenning. Refinement Types for Logical Frameworks. In *TYPES*, pages 285–299, 1993.

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

39 Elaine Pimentel, Simona Ronchi Della Rocca, and Luca Roversi. Intersection types from a proof-theoretic perspective. *Fundam. Inform.*, 121(1-4):253–274, 2012.

40 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.

41 John C. Reynolds. Preliminary design of the programming language Forsythe. Report CMU–CS–88–159, Carnegie Mellon University, 1988.

42 Simona Ronchi Della Rocca and Luca Roversi. Intersection logic. In *CSL*, volume 2142 of *Lecture Notes in Computer Science*, pages 421–428. Springer-Verlag, 2001.

43 Vincent Siles and Hugo Herbelin. Equality is typable in semi-full pure type systems. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS*, pages 21–30, 2010.

44 Claude Stolze. Δ-framework implementation. Bull and Bull-Subtyping, 2017.

45 Claude Stolze, Luigi Liquori, Furio Honsell, and Ivan Scagnetto. Towards a Logical Framework with Intersection and Union Types. In *11th International Workshop on Logical Frameworks and Meta-languages, LFMTP*, pages 1 – 9, 2017.

46 Aaron Stump. From realizability to induction via dependent intersection. *Annals of Pure and Applied Logic*, 169(7):637 – 655, 2018.

47 Betti Venneri. Intersection types as logical formulae. *J. Log. Comput.*, 4(2):109–124, 1994.

48 Joe B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A calculus with polymorphic and polyvariant flow types. *J. Funct. Program.*, 12(3):183–227, 2002.
49 Joe B. Wells and Christian Haack. Branching types. In *ESOP*, volume 2305 of *Lecture Notes in Computer Science*, pages 115–132. Springer-Verlag, 2002.

A Appendix

Let Figure 9 denote Valid Signatures and Contexts and Figure 10 denote Valid Kinds and Families.

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

Let $\Sigma \stackrel{def}{=} \{c_1{:}\sigma_1, \ldots, c_n{:}\sigma_n\}$, and $\Sigma, c{:}\sigma \stackrel{def}{=} \Sigma \cup \{c{:}\sigma\}$

Valid Signatures

$$\frac{}{\langle\rangle \text{ sig}}\ (\epsilon\Sigma) \qquad \frac{\Sigma \text{ sig} \quad \vdash_\Sigma K \quad a \notin \mathrm{dom}(\Sigma)}{\Sigma, a{:}K \text{ sig}}\ (K\Sigma) \qquad \frac{\Sigma \text{ sig} \quad \vdash_\Sigma \sigma : \mathsf{Type} \quad c \notin \mathrm{dom}(\Sigma)}{\Sigma, c{:}\sigma \text{ sig}}\ (\sigma\Sigma)$$

Valid Contexts

$$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \langle\rangle}\ (\epsilon\Gamma) \qquad \frac{\vdash_\Sigma \Gamma \quad \Gamma \vdash_\Sigma \sigma : \mathsf{Type} \quad x \notin \mathrm{dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}\sigma}\ (\sigma\Gamma)$$

Figure 9 Valid Signatures and Contexts

Valid Kinds

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \mathsf{Type}}\ (\mathit{Type}) \qquad \frac{\Gamma, x{:}\sigma \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.K}\ (\Pi K)$$

Valid Families

$$\frac{\vdash_\Sigma \Gamma \quad a{:}K \in \Sigma}{\Gamma \vdash_\Sigma a : K}\ (\mathit{Const}) \qquad \frac{\Gamma \vdash_\Sigma \sigma : K_1 \quad \Gamma \vdash_\Sigma K_2 \quad K_1 =_\Delta K_2}{\Gamma \vdash_\Sigma \sigma : K_2}\ (\mathit{Conv})$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.\tau : \mathsf{Type}}\ (\Pi I) \qquad \frac{\Gamma \vdash_\Sigma \sigma : \Pi x{:}\tau.K \quad \Gamma \vdash_\Sigma \Delta : \tau}{\Gamma \vdash_\Sigma \sigma\,\Delta : K[\Delta/x]}\ (\Pi E)$$

$$\frac{\Gamma \vdash_\Sigma \sigma : \mathsf{Type} \quad \Gamma \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \sigma \to^{\mathsf{r}} \tau : \mathsf{Type}}\ (\to^{\mathsf{r}} I)$$

$$\frac{\Gamma \vdash_\Sigma \sigma : \mathsf{Type} \quad \Gamma \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \sigma \cap \tau : \mathsf{Type}}\ (\cap I) \qquad \frac{\Gamma \vdash_\Sigma \sigma : \mathsf{Type} \quad \Gamma \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \sigma \cup \tau : \mathsf{Type}}\ (\cup I)$$

Figure 10 Valid Kinds and Families

LF$_\Delta$ can play the role of a logical framework only if decidable. The road map which we follow to establish decidability is the standard one, see *e.g.* [22]. In particular, we prove in order: uniqueness of types and kinds, structural properties, normalization for raw well-formed terms, and hence confluence. Then we prove the inversion property, the subderivation

$$
\begin{array}{rcl@{\qquad}rcl}
\|\,\mathsf{Type}\,\| & = & \top \quad \text{(a special constant)} & \|\,\sigma \to^{\mathsf{r}} \tau\,\| & = & \|\,\sigma\,\| \to \|\,\tau\,\| \\
\|\,\Pi x{:}\sigma.K\,\| & = & \|\,\sigma\,\| \to \|\,K\,\| & \|\,\sigma\,\Delta\,\| & = & \|\,\sigma\,\| \\
\|\,a\,\| & = & a & \|\,\sigma \cap \tau\,\| & = & \|\,\sigma\,\| \cap \|\,\tau\,\| \\
\|\,\Pi x{:}\sigma.\tau\,\| & = & \|\,\sigma\,\| \to \|\,\tau\,\| & \|\,\sigma \cup \tau\,\| & = & \|\,\sigma\,\| \cup \|\,\tau\,\|
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
|\,a\,| & = & a & |\,\sigma \to^{\mathsf{r}} \tau\,| & = & c_{\times}\ |\,\sigma\,|\,|\,\tau\,| \\
|\,c\,| & = & c & |\,\sigma \cap \tau\,| & = & c_{\times}\ |\,\sigma\,|\,|\,\tau\,| \\
|\,x\,| & = & x & |\,\sigma \cup \tau\,| & = & c_{\times}\ |\,\sigma\,|\,|\,\tau\,| \\
|\,\sigma\,\Delta\,| & = & |\,\sigma\,|\,|\,\Delta\,| & |\,\langle \Delta_1 \,,\, \Delta_2 \rangle\,| & = & |\,\Delta_1\,| \\
|\,\Delta_1\,\Delta_2\,| & = & |\,\Delta_1\,|\,|\,\Delta_2\,| & |\,[\Delta_1 \,,\, \Delta_2]\,| & = & |\,\Delta_1\,| \\
|\,\Delta_1\,{}^{\mathsf{r}}\Delta_2\,| & = & |\,\Delta_1\,|\,|\,\Delta_2\,| & |\,pr_l\,\Delta\,| & = & |\,\Delta\,| \\
|\,\lambda x{:}\sigma.\Delta\,| & = & (\lambda y.\lambda x.\ |\,\Delta\,|)\ |\,\sigma\,|\ \ y \notin fv(\Delta) & |\,pr_r\,\Delta\,| & = & |\,\Delta\,| \\
|\,\lambda^{\mathsf{r}} x{:}\sigma.\Delta\,| & = & (\lambda y.\lambda x.\ |\,\Delta\,|)\ |\,\sigma\,|\ \ y \notin fv(\Delta) & |\,in_l^{\sigma}\,\Delta\,| & = & (\lambda x.\ |\,\Delta\,|)\ |\,\sigma\,|\ \ x \notin fv(\Delta) \\
|\,\Pi x{:}\sigma.\tau\,| & = & c_{\|\sigma\|}\ |\,\sigma\,|\ (\lambda x.\ |\,\tau\,|) & |\,in_r^{\sigma}\,\Delta\,| & = & (\lambda x.\ |\,\Delta\,|)\ |\,\sigma\,|\ \ x \notin fv(\Delta)
\end{array}
$$

Figure 11 The forgetful mappings $\|\cdot\|$ and $|\cdot|$

property, subject reduction, and finally decidability.

▶ **Lemma 10.** *Let α be either $\sigma : K$ or $\Delta : \sigma$. Then:*

1. *Weakening: If $\Gamma \vdash_\Sigma \alpha$ and $\vdash_\Sigma \Gamma, \Gamma'$, then $\Gamma, \Gamma' \vdash_\Sigma \alpha$.*
2. *Strengthening: If $\Gamma, x{:}\sigma, \Gamma' \vdash_\Sigma \alpha$, then $\Gamma, \Gamma' \vdash_\Sigma \alpha$, provided that $x \notin FV(\Gamma') \cup FV(\alpha)$.*
3. *Transitivity: If $\Gamma \vdash_\Sigma \Delta : \sigma$ and $\Gamma, x{:}\sigma, \Gamma' \vdash_\Sigma \alpha$, then $\Gamma, \Gamma'[\Delta/x] \vdash_\Sigma \alpha[\Delta/x]$.*
4. *Permutation: If $\Gamma, x_1{:}\sigma, \Gamma', x_2{:}\tau, \Gamma'' \vdash_\Sigma \alpha$, then $\Gamma, x_2{:}\tau, \Gamma', x_1{:}\sigma, \Gamma''$ $\vdash_\Sigma \alpha$, provided that x_1 does not occur free in Γ' or in τ, and that τ is valid in Γ.*

▶ **Theorem 3** (Unicity of Types and Kinds)**.**

1. *If $\Gamma \vdash_\Sigma \Delta : \sigma$ and $\Gamma \vdash_\Sigma \Delta : \tau$, then $\sigma =_\Delta \tau$.*
2. *If $\Gamma \vdash_\Sigma \sigma : K$ and $\Gamma \vdash_\Sigma \sigma : K'$, then $K =_\Delta K'$.*

In order to prove strong normalization we follow the pattern used for pure LF. Namely, we map LF$_\Delta$-terms into terms of the system $\mathcal{B}$ in such a way that redexes in the source language are mapped into redexes in the target language, and then take advantage of Theorem 2. Special care is needed in dealing with redexes occurring in type-dependencies, because these need to be flattened at the level of terms.

▶ **Definition 11.** Let the forgetful mappings $\|\cdot\|$ and $|\cdot|$ be defined as in Figure 11.

The forgetful mappings are extended to contexts and signatures in the obvious way. The clauses for strong pairs/co-pairs are justified by the following lemma:

▶ **Lemma 12.** *If $\Gamma \vdash_\Sigma \langle \Delta_1 \,,\, \Delta_2 \rangle : \sigma$ or $\Gamma \vdash_\Sigma [\Delta_1 \,,\, \Delta_2] : \sigma$, then $|\,\Delta_1\,| =_\beta |\,\Delta_2\,|$.*

The following lemmas are proved by straightforward structural induction.

▶ **Lemma 13.**

1. *If $\sigma =_\Delta \tau$, then $\|\,\sigma\,\| =_\beta \|\,\tau\,\|$.*

2. *If* $K_1 =_\Delta K_2$*, then* $\| K_1 \| =_\beta \| K_2 \|$.

▶ **Lemma 14.**
1. $| \Delta_1[\Delta_2/x] | =_\beta | \Delta_1 | \, [| \Delta_2 | \, /x]$.
2. $| \sigma[\Delta/x] | =_\beta | \sigma | \, [| \Delta | \, /x]$.

▶ **Lemma 15.**
1. *If* $\Gamma \vdash_\Sigma \sigma : K$*, then* $\| \Gamma \| \vdash_{\mathcal{B}^+} | \sigma | : \| K \|$.
2. *If* $\Gamma \vdash_\Sigma \Delta : \sigma$*, then* $\| \Gamma \| \vdash_{\mathcal{B}^+} | \Delta | : \| \sigma \|$.
where $\vdash_{\mathcal{B}^+}$ *denotes the type system* $\mathcal{B}$*, augmented by* $c_\times : \top \to \top \to \top$ *and the infinite set of axioms* $c_{\|\sigma\|} : \top \to (\| \sigma \| \to \top) \to \top$*, for each type* σ*.*

Notice that the function $\wr \;\; \wr$ and $| \;\; |$ treat differently relevant implication.

▶ **Lemma 16.**
1. *If* $\sigma \longrightarrow_\beta \tau$*, then* $| \sigma | \longrightarrow^+_\beta | \tau |$.
2. *if* $\Delta_1 \longrightarrow_\beta \Delta_2$*, then* $| \Delta_1 | \longrightarrow^+_\beta | \Delta_2 |$.

Parallel reduction enjoys the strong normalization property, *i.e.*

▶ **Theorem 4** (Strong normalization)**.**
1. *The* LF_Δ *is strongly normalizing, i.e.,*
  a. *If* $\Gamma \vdash_\Sigma K$*, then* K *is strongly normalizing.*
  b. *If* $\Gamma \vdash_\Sigma \sigma : K$*, then* σ *is strongly normalizing.*
  c. *If* $\Gamma \vdash_\Sigma \Delta : \sigma$*, then* Δ *is strongly normalizing.*
2. *Every strongly normalizing pure* λ*-term can be annotated so as to be the essence of a* Δ*-term.*

Proof. 1) Strong normalization derives directly from Lemmas 15, 16 and Theorem 2.
2) By induction on the specification of strongly normalizing terms which can be inductively defined as i) $\Delta_1 \ldots \Delta_n \in SN \Rightarrow \lambda x_1, \ldots, x_n.x\, \Delta_1 \ldots \Delta_n \in SN$ for x possibly among the x_i's, ii) $\Delta[\Delta'/x]\, \Delta_1 \ldots \Delta_n \in SN$, and iii) $\Delta' \in SN \Rightarrow (\lambda x{:}\sigma.\Delta)\, \Delta'\, \Delta_1 \,\ldots\, \Delta_n \in SN$. ◀

Local confluence (Proposition 1) and strong normalization (Theorem 4) entail confluence, so we have

▶ **Theorem 5** (Confluence)**.** LF_Δ *is confluent, i.e.:*
1. *If* $K_1 \longrightarrow^*_\Delta K_2$ *and* $K_1 \longrightarrow^*_\Delta K_3$*, then* $\exists K_4$ *such that* $K_2 \longrightarrow^*_\Delta K_4$ *and* $K_3 \longrightarrow^*_\Delta K_4$*.*
2. *If* $\sigma_1 \longrightarrow^*_\Delta \sigma_2$ *and* $\sigma_1 \longrightarrow^*_\Delta \sigma_3$*, then* $\exists \sigma_4$ *such that* $\sigma_2 \longrightarrow^*_\Delta \sigma_4$ *and* $\sigma_3 \longrightarrow^*_\Delta \sigma_4$*.*
3. *If* $\Delta_1 \longrightarrow^*_\Delta \Delta_2$ *and* $\Delta_1 \longrightarrow^*_\Delta \Delta_3$*, then* $\exists \Delta_4$ *such that* $\Delta_2 \longrightarrow^*_\Delta \Delta_4$ *and* $\Delta_3 \longrightarrow^*_\Delta \Delta_4$*.*

The following lemmas are proved by structural induction.

▶ **Lemma 17** (Inversion properties)**.**
1. *If* $\Pi x{:}\sigma.\tau =_\Delta \tau''$*, then* $\tau'' \equiv \Pi x{:}\sigma'.\tau'$*, for some* σ', τ'*, such that* $\sigma' =_\Delta \sigma$*, and* $\tau' =_\Delta \tau$*.*
2. *If* $\sigma \to^{\mathsf{r}} \tau =_\Delta \tau''$*, then* $\tau'' \equiv \sigma' \to^{\mathsf{r}} \tau'$*, for some* σ', τ'*, such that* $\sigma' =_\Delta \sigma$*, and* $\tau' =_\Delta \tau$*.*
3. *If* $\sigma \cap \tau =_\Delta \rho$*, then* $\rho \equiv \sigma' \cap \tau'$*, for some* σ', τ'*, such that* $\sigma' =_\Delta \sigma$*, and* $\tau' =_\Delta \tau$*.*
4. *If* $\sigma \cup \tau =_\Delta \rho$*, then* $\rho \equiv \sigma' \cup \tau'$*, for some* σ', τ'*, such that* $\sigma' =_\Delta \sigma$*, and* $\tau' =_\Delta \tau$*.*
5. *If* $\Gamma \vdash_\Sigma \lambda x{:}\sigma.\Delta : \Pi x{:}\sigma.\tau$*, then* $\Gamma, x{:}\sigma \vdash_\Sigma \Delta : \tau$*.*
6. *If* $\Gamma \vdash_\Sigma \lambda\!\!\!/ x{:}\sigma.\Delta : \Pi x{:}\sigma.\tau$*, then* $\Gamma, x{:}\sigma \vdash_\Sigma \Delta : \tau$ *and* $\wr \Delta \wr =_\eta x$*.*
7. *If* $\Gamma \vdash_\Sigma \langle \Delta_1 \,,\, \Delta_2 \rangle : \sigma \cap \tau$*, then* $\Gamma \vdash_\Sigma \Delta_1 : \sigma$*,* $\Gamma \vdash_\Sigma \Delta_2 : \tau$*, and* $\wr \Delta_1 \wr =_\beta \wr \Delta_2 \wr$*.*
8. *If* $\Gamma \vdash_\Sigma [\Delta_1 \,,\, \Delta_2] : \Pi x{:}\sigma \cup \tau.\rho$*, then* $\Gamma \vdash_\Sigma \Delta_1 : \Pi y{:}\sigma.\rho\,(in^\tau_l\ y)$*,* $\Gamma \vdash_\Sigma \Delta_2 : \Pi y{:}\tau.\rho\,(in^\sigma_r\ y)$*, and* $\wr \Delta_1 \wr =_\beta \wr \Delta_2 \wr$*.*

9. *If* $\Gamma \vdash_\Sigma pr_l\,\Delta : \sigma$*, then* $\Gamma \vdash_\Sigma \Delta{:}\sigma \cap \tau$*, for some* τ*.*
10. *If* $\Gamma \vdash_\Sigma pr_r\,\Delta : \tau$*, then* $\Gamma \vdash_\Sigma \Delta{:}\sigma \cap \tau$*, for some* σ*.*
11. *If* $\Gamma \vdash_\Sigma in_l^\tau\,\Delta : \sigma \cup \tau$*, then* $\Gamma \vdash_\Sigma \Delta : \sigma$ *and* $\Gamma \vdash_\Sigma \sigma \cup \tau : \mathsf{Type}$*.*
12. *If* $\Gamma \vdash_\Sigma in_r^\sigma\,\Delta : \sigma \cup \tau$*, then* $\Gamma \vdash_\Sigma \Delta : \tau$ *and* $\Gamma \vdash_\Sigma \sigma \cup \tau : \mathsf{Type}$*.*

▶ **Proposition 18** (Subderivation)**.**
1. *A derivation of* $\vdash_\Sigma \langle\rangle$ *has a subderivation of* Σ *sig.*
2. *A derivation of* $\Sigma, a{:}K$ *sig has subderivations of* Σ *sig and* $\vdash_\Sigma K$*.*
3. *A derivation of* $\Sigma, f{:}\sigma$ *sig has subderivations of* Σ *sig and* $\vdash_\Sigma \sigma{:}\mathsf{Type}$*.*
4. *A derivation of* $\vdash_\Sigma \Gamma, x{:}\sigma$ *has subderivations of* Σ *sig,* $\vdash_\Sigma \Gamma$*, and* $\Gamma \vdash_\Sigma \sigma{:}\mathsf{Type}$*.*
5. *A derivation of* $\Gamma \vdash_\Sigma \alpha$ *has subderivations of* Σ *sig and* $\vdash_\Sigma \Gamma$*.*
6. *Given a derivation of the judgement* $\Gamma \vdash_\Sigma \alpha$*, and a subterm occurring in the subject of this judgement, there exists a derivation of a judgement having this subterm as a subject.*

▶ **Theorem 6** (Subject reduction of LF_Δ)**.**
1. *If* $\Gamma \vdash_\Sigma K$*, and* $K \to_\Delta K'$*, then* $\Gamma \vdash_\Sigma K'$*.*
2. *If* $\Gamma \vdash_\Sigma \sigma : K$*, and* $\sigma \to_\Delta \sigma'$*, then* $\Gamma \vdash_\Sigma \sigma' : K$*.*
3. *If* $\Gamma \vdash_\Sigma \Delta : \sigma$*, and* $\Delta \to_\Delta \Delta'$*, then* $\Gamma \vdash_\Sigma \Delta' : \sigma$*.*

Finally, we define a possible algorithm for checking judgements in LF_Δ by computing a type or a kind for a term, and then testing for *definitional equality*, *i.e.* $=_\Delta$, against the given type or kind. This is achieved by reducing both to their unique normal forms and checking that they are identical up to α-conversion. Therefore we finally have:

▶ **Theorem 7** (Decidability)**.** *All the type judgments of* LF_Δ *are recursively decidable.*

Minimal Relevant Implications and Type Inclusion. Type inclusion and the rules of *subtyping* are related to the notion of minimal relevant implication, see [4, 17]. The insight is quite subtle, but ultimately very simple. This is what makes it appealing. The apparently intricate rules of subtyping and type inclusion, which occur in many systems, and might even appear *ad hoc* at times, can all be explained away in our principled approach, by proving that the relevant implication type is inhabited by a term whose essence is essentially a variable.

The following theorem we show how relevant implication subsumes the type-inclusion rules of the theory Ξ of [3], without rule (10): we call Ξ' the resulting set.

▶ **Theorem 8** (Type Inclusion)**.** *The judgement* $\langle\rangle \vdash_\Sigma \Delta : \sigma \to^{\mathsf{r}} \tau$ *(where both* σ *and* τ *do not contain dependencies or relevant families) holds iff* $\sigma \leq \tau$ *holds in the subtype theory* Ξ' *of* $\mathcal{B}$ *enriched with new axioms of the form* $\sigma_1 \leq \sigma_2$ *for each constant* $c : \sigma_1 \to^{\mathsf{r}} \sigma_2 \in \Sigma$*.*

Proof.

(if). Follows directly from Lemma 17.

(only if). It is possible to write a Δ-term whose essence is an $\eta-$expansion of the identity $(\lambda x.x)$ corresponding to each of the axioms and rules in Ξ'. The Δ-term is obtained by defining a function $\|\sigma \leq \tau\|_\Delta$, where $\sigma \leq \tau$ is a subtyping derivation tree in the type theory Ξ', which coerce a Δ-term from type σ to type τ:

$$
\begin{array}{lrcl}
(1) & \|\sigma \leqslant \sigma \cap \sigma\|_{\Delta} & \stackrel{def}{=} & \langle \Delta \ , \ \Delta \rangle \\
(2) & \|\sigma \cup \sigma \leqslant \sigma\|_{\Delta} & \stackrel{def}{=} & [\lambda x{:}\sigma.x \ , \ \lambda x{:}\sigma.x]\,\Delta \\
(3) & \|\sigma_1 \cap \sigma_2 \leqslant \sigma_i\|_{\Delta} & \stackrel{def}{=} & pr_i\,\Delta \\
(4) & \|\sigma_i \leqslant \sigma_1 \cup \sigma_2\|_{\Delta} & \stackrel{def}{=} & in_i\,\Delta \\
(6) & \|\sigma \leqslant \sigma\|_{\Delta} & \stackrel{def}{=} & \Delta \\
(7) & \left\|\dfrac{\sigma_1 \leqslant \sigma_2 \quad \tau_1 \leqslant \tau_2}{\sigma_1 \cap \tau_1 \leqslant \sigma_2 \cap \tau_2}\right\|_{\Delta} & \stackrel{def}{=} & \langle \|\sigma_1 \leqslant \sigma_2\|_{(pr_l\,\Delta)} \ , \ \|\tau_1 \leqslant \tau_2\|_{(pr_r\,\Delta)} \rangle \\
(8) & \left\|\dfrac{\sigma_1 \leqslant \sigma_2 \quad \tau_1 \leqslant \tau_2}{\sigma_1 \cup \tau_1 \leqslant \sigma_2 \cup \tau_2}\right\|_{\Delta} & \stackrel{def}{=} & [\lambda x{:}\sigma_1.in_l^{\tau_2}\ \|\sigma_1 \leqslant \sigma_2\|_x \ , \ \lambda x{:}\tau_1.in_r^{\sigma_2}\ \|\tau_1 \leqslant \tau_2\|_x]\,\Delta \\
(9) & \left\|\dfrac{\sigma \leqslant \tau \quad \tau \leqslant \rho}{\sigma \leqslant \rho}\right\|_{\Delta} & \stackrel{def}{=} & \|\tau \leqslant \rho\|_{(\|\sigma \leqslant \tau\|_{\Delta})} \\
(11) & \|(\sigma \to \tau) \cap (\sigma \to \rho) \leqslant \sigma \to (\tau \cap \rho)\|_{\Delta} & \stackrel{def}{=} & \lambda x{:}\sigma.\langle (pr_l\,\Delta)\,x \ , \ (pr_r\,\Delta)\,x \rangle \\
(12) & \|(\sigma \to \rho) \cap (\tau \to \rho) \leqslant (\sigma \cup \tau) \to \rho\|_{\Delta} & \stackrel{def}{=} & \lambda x{:}\sigma \cup \tau.[\lambda y{:}\sigma.(pr_l\,\Delta)\,y \ , \ \lambda y{:}\tau.(pr_r\,\Delta)\,y]\,x \\
(14) & \left\|\dfrac{\sigma_2 \leqslant \sigma_1 \quad \tau_1 \leqslant \tau_2}{\sigma_1 \to \tau_1 \leqslant \sigma_2 \to \tau_2}\right\|_{\Delta} & \stackrel{def}{=} & \lambda x{:}\sigma_2.\,\|\tau_1 \leqslant \tau_2\|_{(\Delta\ \|\sigma_2 \leqslant \sigma_1\|_x)}
\end{array}
$$

◀

A.1 Typed derivation of Pierce's example of Subsection 2.1

$$
\cfrac{\cfrac{\cfrac{\begin{array}{l}
\Gamma \vdash_\Sigma \lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1 : \Pi x_1{:}\sigma_1.(a\, x_4\, x_4)[in_l^{\sigma_2}\, x_1/x_4] \\
\Gamma \vdash_\Sigma \lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2 : \Pi x_2{:}\sigma_2.(a\, x_4\, x_4)[in_r^{\sigma_1}\, x_2/x_4] \\
\Gamma, x_4{:}\sigma_1 \cup \sigma_2 \vdash_\Sigma a\, x_4\, x_4 : \mathsf{Type} \\
\wr\, \lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1 \,\wr =_\eta \wr\, \lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2)\, \wr
\end{array}}{\Gamma \vdash_\Sigma [\lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1 \ ,\ \lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2] : \Pi x_4{:}\sigma_1 \cup \sigma_2.a\, x_4\, x_4}\ (\cup E) \quad \Gamma \vdash_\Sigma (\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)\, y\, z : \sigma_1 \cup \sigma_2}{\Gamma \vdash_\Sigma [(\lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1)\ ,\ (\lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2)]\, ((\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)\, y\, z) : a\, ((\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)\, y\, z)\, ((\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)\, y\, z)}\ (\Pi E)}{\Gamma \vdash_\Sigma [\lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1)\ ,\ \lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2)]\, ((\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)\, y\, z) : a\, (y\, z)\, (y\, z)}\ (Conv)
$$

where

$$\Gamma \stackrel{def}{=} x{:}\Pi x_1{:}\sigma_1.\Pi x_2{:}\sigma_1.a\,(in_l^{\sigma_2}\ x_1)\,(in_l^{\sigma_2}\ x_2) \cap \Pi x_1{:}\sigma_2.\Pi x_2{:}\sigma_2.a\,(in_r^{\sigma_1}\ x_1)\,(in_r^{\sigma_1}\ x_2), y{:}\rho \to \sigma_1 \cup \sigma_2, z{:}\rho$$

and

$$\Sigma \stackrel{def}{=} a{:}\sigma_1 \cup \sigma_2 \to \sigma_1 \cup \sigma_2 \to \mathsf{Type}$$