$\mathsf{LF}_{\mathcal{P}}$ – A Logical Framework with External Predicates

Furio Honsell
Università di Udine, Italy
furio.honsell@uniud.it

Marina Lenisa
Università di Udine, Italy
marina.lenisa@uniud.it

Luigi Liquori
Institut National de Recherche en Informatique et en Automatique, France
Luigi.Liquori@inria.fr

Petar Maksimovic
Université de Nice Sophia-Antipolis, France,
Mathematical Institute of the Serbian Academy of Sciences and Arts, Serbia
petarmax@mi.sanu.ac.rs

Ivan Scagnetto
Università di Udine, Italy
ivan.scagnetto@uniud.it

Abstract

The $\mathsf{LF}_{\mathcal{P}}$ Framework is an extension of the Harper-Honsell-Plotkin's Edinburgh Logical Framework LF with *external predicates*. This is accomplished by defining *lock type constructors*, which are a sort of $\diamond$-*modality constructors*, releasing their argument *under the condition* that a possibly *external predicate* is satisfied on an appropriate typed judgement. Lock types are defined using the standard pattern of constructive type theory, *i.e.* via *introduction*, *elimination*, and *equality rules*. Using $\mathsf{LF}_{\mathcal{P}}$, one can factor out the complexity of encoding specific features of logical systems which are awkwardly encoded in LF, *e.g.* side-conditions in the application of rules in Modal Logics, substructural rules as in *non-commutative Linear Logic*, and pre- and post-conditions in Hoare-like programming languages. Once these conditions have been isolated, their *verification* can be delegated to an external proof engine, in the style of *Poincaré Principle*. We investigate and characterize the metatheoretical properties of the calculus underpinning $\mathsf{LF}_{\mathcal{P}}$, proving strong normalization, confluence, and subject reduction. This latter property holds under the assumption that predicates are *well-behaved*, *i.e. closed under weakening, permutation*, *substitution*, and $\beta\mathcal{L}$*-reduction* in the arguments.

Categories and Subject Descriptors F.3.1 [*Specifying and Verifying and Reasoning about Programs*]: Mechanical verification

General Terms Theory, Verification

Keywords Type theory, Logical Frameworks

1. Introduction

The Edinburgh Logical Framework LF of [11] is a first-order constructive type theory. It was introduced as a *general metalanguage for logics* as well as a specification language for *generic proof-development environments*. In this paper, we consider an extension of LF with *external predicates*. This is accomplished by defining *lock type constructors*, which are a sort of $\diamond$-*modality constructors* for building types of the shape $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, where $\mathcal{P}$ is a predicate on type judgements.

Following the standard specification paradigm in Constructive Type Theory, we define lock types using *introduction*, *elimination*, and *equality rules*. Namely, we introduce a lock *constructor* for building objects $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]$ of type $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, via the *introduction rule* (I) below. Correspondingly, we introduce an unlock *destructor*, $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]$, and an *elimination rule* (E) which allows for the elimination of the lock type constructor, under the condition that a specified predicate $\mathcal{P}$ is verified, possibly *externally*, on an appropriate *correct*, *i.e.* derivable, judgement.

$$\frac{\Gamma \vdash_\Sigma M : \rho \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]}\,(\mathsf{I}) \qquad \frac{\begin{array}{c}\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \\ \Gamma \vdash_\Sigma N : \sigma \quad \mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)\end{array}}{\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho}\,(\mathsf{E})$$

The *equality rule* for lock types amounts to a lock reduction ($\mathcal{L}$-reduction), $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\mathcal{L}} M$, which allows the elimination of a *lock*, in the presence of an *unlock*. The $\mathcal{L}$-reduction combines with standard β-reduction into $\beta\mathcal{L}$-reduction.

$\mathsf{LF}_{\mathcal{P}}$ is parametric over a set of (*well-behaved*) predicates $\mathcal{P}$, which are defined on derivable typing judgements of the form $\Gamma \vdash_\Sigma N : \sigma$. The syntax of $\mathsf{LF}_{\mathcal{P}}$ predicates is not specified, with the idea being that their truth is verified via an *external call* to a logical system; one can view this externalization as an *oracle call*. Thus, $\mathsf{LF}_{\mathcal{P}}$ allows for the invocation of external "modules" which, in principle, can be executed elsewhere, and whose successful verification can be acknowledged in the system via $\mathcal{L}$-reduction. Pragmatically, lock types allow for the factoring out of the complexity of derivations by delegating the {verification, computation} of such predicates to an external proof engine or tool. Proof terms do not contain explicit evidence for external predicates, but just record that a verification has {to be, been} carried out. Thus, we combine the reliability of formal proof systems based on constructive type theory with the efficiency of other computer tools, in the style of the *Poincaré Principle* [4].

In this paper, we develop the metatheory of $\mathsf{LF}_{\mathcal{P}}$. Strong normalization and confluence are proven without any assumptions on predicates. For subject reduction, we require the predicates to be *well-behaved*, *i.e. closed under weakening, permutation*, *substitu-*

tion, and *$\beta\mathcal{L}$-reduction* in the arguments. $\mathsf{LF}_{\mathcal{P}}$ is decidable, if the external predicates are decidable.

Moreover, we sketch a *library* of external predicates, which we use to present significant examples of encodings in $\mathsf{LF}_{\mathcal{P}}$, which are awkward in LF. In particular, we give smooth encodings of side conditions in the rules of Modal Logics, in Natural Deduction style, *cf.* [2, 8]. We also encode substructural logics, including non-commutative Linear Logic, *cf.* [8, 21]. $\mathsf{LF}_{\mathcal{P}}$ further supports natural dealing with *program correctness* and Hoare-like logics. Capitalizing on the call to external logical systems via a simple term application, $\mathsf{LF}_{\mathcal{P}}$ greatly simplifies the task of embedding pre- and post-conditions in programming languages, providing a smooth way for bridging the gap between proof assistants and prototype programming languages. Our approach, via oracles, is external (cf. [10] for a different, "internal" approach).

As far as expressivity is concerned, $\mathsf{LF}_{\mathcal{P}}$ is a stepping stone towards a general theory of *shallow vs. deep encodings*, with our encodings being shallow by definition. Clearly, by Church's thesis, all external decidable predicates in $\mathsf{LF}_{\mathcal{P}}$ can be encoded, possibly with very deep encodings, in standard LF. It would be interesting to state in a precise categorical setting the relationship between such deep internal encodings and the encodings in $\mathsf{LF}_{\mathcal{P}}$. $\mathsf{LF}_{\mathcal{P}}$ can be viewed also as a neat methodology for separating the logical contents from the verification, often cumbersome but ultimately computable, of structural and syntactical properties.

Comparison with related work. The present paper continues the research line of [13, 14], which present extensions of the original Logical Framework LF, where a notion of β-reduction *modulo* a predicate $\mathcal{P}$ is considered. These capitalize on the idea of *stuck-reductions* in objects and types in the setting of higher-order term rewriting systems, by Cirstea-Kirchner-Liquori [5, 7]. In [13, 14] the dependent function type is conditioned by a predicate, and we have a corresponding *conditioned* β-reduction, which fires when the predicate holds on a $\{$term, judgement$\}$. In $\mathsf{LF}_{\mathcal{P}}$, predicates are external to the system and the verification of the validity of the predicate is part of the typing system. Standard β-reduction is recovered and combined with an *unconditioned* lock reduction. The move of having predicates as new type constructors rather than as parameters of Π's and λ's allows $\mathsf{LF}_{\mathcal{P}}$ to be a mere *language extension* of standard LF. This simplifies the metatheory, while providing a more modular approach.

Our approach generalizes and subsumes, in an abstract way, other approaches in the literature, which combine internal and external derivations. And in many cases it can express and incorporate these alternate approaches. The relationships with the systems of [5, 7, 13, 14], which combine derivation and computation, have been discussed above. Systems supporting the *Poincaré Principle* [4], or *Deduction Modulo* [9], where derivation is separated from verification, can be directly incorporated in $\mathsf{LF}_{\mathcal{P}}$. Similarly, we can abstractly subsume the system presented in [6], which addresses a specific instance of our problem: how to outsource the computation of a decision procedure in Type Theory in a sound and principled way via an abstract conversion rule.

The work presented here also has a bearing on proof irrelevance. In [18], two terms inhabiting the same *proof irrelevant type* are set to be equal. However, when dealing with proof irrelevance in this way, a great amount of internal work is required, all of the relevant rules have to be explicitly specified in the signature, in that the *irrelevant* terms need to be derived in the system anyway. With our approach, we move one step further, and we do away completely with *irrelevant* terms in the system by simply delegating the task of building them to the external proof verifier. We limit ourselves, in $\mathsf{LF}_{\mathcal{P}}$, to the recording, through a lock type, that one such evidence, possibly established somewhere else, needs to be provided, making our approach more modular.

In the present work, predicates are defined on derivable judgements, and hence may, in particular, inspect the signature and the context, which normal LF cannot. The ability to inspect the signature and the context is reminiscent of [19, 20], although in that approach the inspection was layered upon LF; in $\mathsf{LF}_{\mathcal{P}}$ it is integrated in the system. This integration is closer to the approach of [16], but more work needs to be done to compare precisely the expressive powers.

Another interesting framework, which adds a layer on top of LF is the Delphin system [22], providing a functional programming language allowing the user to encode, manipulate, and reason over dependent higher-order datatypes. However, also in this case the focus is at the computational level inside the framework, rather than at the capability of delegating the verification of predicates to an external oracle.

LF with Side Conditions (LFSC), presented in [23], is more reminiscent of our approach since "it extends LF to allow side conditions to be expressed using a simple first-order functional programming language". Indeed, the author aims at factoring out of the main proof the verifications of (complicated) side-conditions. Such task is delegated to the type checker which runs the code associated with the side-condition, verifying that it yields the expected output. The proposed machinery is focused on providing improvements for solvers related to Satisfiability Modulo Theories (SMT).

Synopsis. In Section 2, we present the syntax of $\mathsf{LF}_{\mathcal{P}}$, the typing system, and the $\beta\mathcal{L}$-reduction, together with the main metatheoretical properties of the system. In Section 3, we show how to encode call-by-value λ-calculus, Modal Logics, and non-commutative Linear Logic. Conclusions and future work appear in Section 4. In Appendix A, we collect complete definitions and proofs of the properties of $\mathsf{LF}_{\mathcal{P}}$, and proofs of the adequacy results.

An extended version of the present paper, including a canonical version of $\mathsf{LF}_{\mathcal{P}}$ and more examples, appears in [15].

2. The Framework

The pseudo-syntax of $\mathsf{LF}_{\mathcal{P}}$ is presented in Figure 1. It is essentially that of LF, with the addition, on families and objects, of a *lock constructor*, $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[-]$, and a corresponding *lock destructor*, $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[-]$, on objects, both parametrized over a logical predicate $\mathcal{P}$. The predicate $\mathcal{P}$ ranges over a set of unary predicates, defined on derivable type judgements of the form $\Gamma \vdash_\Sigma N : \sigma$. $\mathsf{LF}_{\mathcal{P}}$ is parametric over a finite set of such predicates, the syntax of which, as they are external, is not specified. However, these predicates have to satisfy certain conditions, which will be discussed below, in order to ensure subject reduction of the system.

Notational conventions and auxiliary definitions. Let T range over any term of the calculus (kind, family, object). Let the symbol $\equiv$ denote syntactic identity on terms. The domain $\mathsf{Dom}(\Gamma)$ is defined as usual. The definitions of free and bound variables, as well as substitution are naturally extended for locked and unlocked types and objects. In particular, a substitution $[M/x]$ on a term $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[T]$ affects T, N, and σ, *i.e.* $(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[T])[M/x] = \mathcal{L}^{\mathcal{P}}_{N[M/x],\sigma[M/x]}[T[M/x]]$, and similarly for terms with the lock destructor. As usual, we suppose that, in the context $\Gamma, x{:}\sigma$, the variable x does not occur free in Γ or in σ. We will work modulo α-conversion and Barendregt's hygiene condition. All of the symbols can appear indexed.

The type system for $\mathsf{LF}_{\mathcal{P}}$ proves judgements of the shape:

Σ	sig		Σ is a valid signature
	$\vdash_\Sigma$	Γ	Γ is a valid context in Σ
Γ	$\vdash_\Sigma$	K	K is a kind in Γ and Σ
Γ	$\vdash_\Sigma$	$\sigma : K$	σ has kind K in Γ and Σ
Γ	$\vdash_\Sigma$	$M : \sigma$	M has type σ in Γ and Σ

$$
\begin{array}{llcll}
\Sigma \in \mathcal{S} & \Sigma & ::= & \emptyset \mid \Sigma, a{:}K \mid \Sigma, c{:}\sigma & \textit{Signatures} \\
\Gamma \in \mathcal{C} & \Gamma & ::= & \emptyset \mid \Gamma, x{:}\sigma & \textit{Contexts} \\
K \in \mathcal{K} & K & ::= & \mathsf{Type} \mid \Pi x{:}\sigma.K & \textit{Kinds} \\
\sigma, \tau, \rho \in \mathcal{F} & \sigma & ::= & a \mid \Pi x{:}\sigma.\tau \mid \sigma\, N \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] & \textit{Families} \\
M, N \in \mathcal{O} & M & ::= & c \mid x \mid \lambda x{:}\sigma.M \mid M\, N \mid & \\
 & & & \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \mid \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] & \textit{Objects}
\end{array}
$$

Figure 1. $\mathsf{LF}_{\mathcal{P}}$ Syntax

$$
\begin{array}{ll}
(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x] & (\beta{\cdot}\mathit{Main}) \\
\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} M & (\mathcal{L}{\cdot}\mathit{Main})
\end{array}
$$

Figure 3. Main one-step-$\beta\mathcal{L}$-reduction rules in $\mathsf{LF}_{\mathcal{P}}$

In a typing judgement $\Gamma \vdash_\Sigma T : T'$ (resp. $\Gamma \vdash_\Sigma T$), T will be referred to as the *subject* of that judgement. The typing rules of $\mathsf{LF}_{\mathcal{P}}$ are presented in Figure 2. The rule $(F{\cdot}Lock)$ is used to form a lock type; the rule $(O{\cdot}Lock)$ is the corresponding introduction rule for building objects of the lock type, while the rule $(O{\cdot}Unlock)$ is the elimination rule. It applies only when the predicate $\mathcal{P}$ holds.

In $\mathsf{LF}_{\mathcal{P}}$, we will have two types of reduction: standard β-reduction and $\mathcal{L}$-reduction. The latter allows to dissolve a lock, in presence of a lock destructor (see Figure 3 for the main $\beta\mathcal{L}$-reduction rules on "raw terms", and Figures 4–9 for the contextual closure and $\beta\mathcal{L}$-equivalence).

Here, we present the main properties of $\mathsf{LF}_{\mathcal{P}}$ (details and proofs appear in Appendix A). Without any additional assumptions concerning predicates, the type system is strongly normalizing and confluent. The former follows from the strong normalization result for LF (see [11]), while the latter follows from strong normalization and local confluence, using Newman's Lemma. Weakening and Permutation can be proven under the assumption that the predicates are *closed under weakening* and *permutation of the signature and context*, while Transitivity can be proven under the extra assumption that the predicates are closed under substitution in the argument (*closure under substitution*). For Subject Reduction, we also require the predicates to be closed under $\beta\mathcal{L}$-reduction in the argument (*closure under reduction*). All of the above conditions on predicates are collected in the definition of *well-behaved predicates*:

Definition 1 (Well-behaved predicates)**.** *A finite set of predicates* $\{\mathcal{P}_i\}_{i\in I}$ *is* well-behaved *if each* $\mathcal{P}$ *in the set satisfies the following conditions:*

Closure under signature, context weakening and permutation. *If* Σ *and* Ω *are valid signatures with every declaration in* Σ *also occuring in* Ω*, and* Γ *and* Δ *are valid contexts with every declaration in* Γ *also occuring in* Δ*, and* $\mathcal{P}(\Gamma \vdash_\Sigma \alpha)$ *holds, then* $\mathcal{P}(\Delta \vdash_\Omega \alpha)$ *also holds.*

Closure under substitution. *If* $\mathcal{P}(\Gamma, x{:}\sigma', \Gamma' \vdash_\Sigma N : \sigma)$ *holds, and* $\Gamma \vdash_\Sigma N' : \sigma'$*, then* $\mathcal{P}(\Gamma, \Gamma'[N'/x] \vdash_\Sigma N[N'/x] : \sigma[N'/x])$ *also holds.*

Closure under reduction. *If* $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$ *holds and* $N \to_{\beta\mathcal{L}} N'$ *(resp.* $\sigma \to_{\beta\mathcal{L}} \sigma'$*) holds, then* $\mathcal{P}(\Gamma \vdash_\Sigma N' : \sigma)$ *(resp.* $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma')$*) also holds.*

Theorem 1. *In* $\mathsf{LF}_{\mathcal{P}}$*, the following properties hold:*

Strong normalization.

1. *If* $\Gamma \vdash_\Sigma K$*, then* K *is* $\beta\mathcal{L}$*-strongly normalizing.*
2. *if* $\Gamma \vdash_\Sigma \sigma : K$*, then* σ *is* $\beta\mathcal{L}$*-strongly normalizing.*
3. *if* $\Gamma \vdash_\Sigma M : \sigma$*, then* M *is* $\beta\mathcal{L}$*-strongly normalizing.*

Confluence. $\beta\mathcal{L}$*-reduction is confluent: if* $T \twoheadrightarrow_{\beta\mathcal{L}} T'$ *and* $T \twoheadrightarrow_{\beta\mathcal{L}} T''$*, then there exists* T''' *such that* $T' \twoheadrightarrow_{\beta\mathcal{L}} T'''$ *and* $T'' \twoheadrightarrow_{\beta\mathcal{L}} T'''$*.*

Transitivity. *If predicates are well-behaved, then: if* $\Gamma, x{:}\sigma, \Gamma' \vdash_\Sigma \alpha$*, and* $\Gamma \vdash_\Sigma N : \sigma$*, then* $\Gamma, \Gamma'[N/x] \vdash_\Sigma \alpha[N/x]$*.*

Subject reduction. *If predicates are well-behaved, then:*

1. *If* $\Gamma \vdash_\Sigma K$ *and* $K \to_{\beta\mathcal{L}} K'$*, then* $\Gamma \vdash_\Sigma K'$*.*
2. *If* $\Gamma \vdash_\Sigma \sigma : K$ *and* $\sigma \to_{\beta\mathcal{L}} \sigma'$*, then* $\Gamma \vdash_\Sigma \sigma' : K$*.*
3. *If* $\Gamma \vdash_\Sigma M : \sigma$ *and* $M \to_{\beta\mathcal{L}} M'$*, then* $\Gamma \vdash_\Sigma M' : \sigma$*.*

2.1 The expressive power of $\mathsf{LF}_{\mathcal{P}}$

Various natural questions arise as to the expressive power of $\mathsf{LF}_{\mathcal{P}}$. In this subsection, we only outline answers to some of these questions.

- $\mathsf{LF}_{\mathcal{P}}$ is decidable, if the predicates are decidable; this can be proven as usual.
- If a predicate is *definable in* LF, *i.e.* it can be encoded via the inhabitability of a suitable LF dependent type, then it is well-behaved in the sense of Definition 1.
- All well-behaved r.e. predicates are LF-definable by Church's thesis. Of course, the issue is then on how "deep" the encoding is. To give a more precise answer, we would need a more accurate definition of "deep" and "shallow" encodings, which we still lack. This paper can be seen as a stepping stone towards such a theory, with our approach being "shallow" by definition, and the encodings via Church's thesis being potentially very, very deep.
- One may ask what the relation is between the LF encodings of, say, Modal Logics, which are discussed in [2, 8], and the encodings which appear in this paper (see Section 3.2 below). The former essentially correspond to the internal encoding of the predicates that are utilized in Section 3.2. In fact, one could express the mapping between the two signatures as a forgetful functor going from $\mathsf{LF}_{\mathcal{P}}$ judgements to LF judgements.
- Notice that, even when restricted to *closed normal forms*, so as to be closed under substitution and reduction, well-behaved predicates cannot be naturally encoded in pure LF. *E.g.* only an infinite signature would allow an immediate encoding in LF of the well-behaved predicate "M, N are two different closed normal forms".
- In order to deal in $\mathsf{LF}_{\mathcal{P}}$ with decidable predicates on *open* terms, we need to introduce, as in Section 3.2, suitable constants together with some auxiliary predicates, *e.g* non-occurrence of a constant or closedeness.
- Finally, we can say that, as far as decidable predicates, $\mathsf{LF}_{\mathcal{P}}$ is morally a *conservative extension* of LF. Of course, pragmatically, it is very different, in that it allows for neat factoring out of the true logical contents of derivations from the mere effective verification of other, *e.g.* syntactical or structural properties. A feature of our approach is that of making explicit such a separation.
- The main advantage of having externally verified predicates amount to a smoother encoding (the signature is not cluttered by auxiliary notions and mechanisms needed to implement the predicate), allowing for the optimization of performance, if the external system used to encode the predicate is an optimized tool, specifically designed for the issue at hand (*e.g.* analytic tableaux methods for propositional formulæ).

3. Pragmatics and Case Studies

In this section, we illustrate the pragmatics of using $\mathsf{LF}_{\mathcal{P}}$ as a metalanguage by encoding some crucial case studies.

We focus on formal systems where derivation rules are subject to *side conditions* which are either rather difficult or impossible to encode naively in a type theory-based LF, due to limitations of the latter or to the fact that they need to access the derivation

Signature rules

$$\frac{}{\emptyset\ \mathsf{sig}}\ (\mathsf{S{\cdot}Empty})$$

$$\frac{\begin{array}{l}\Sigma\ \mathsf{sig}\\ \vdash_\Sigma K \quad a \notin \mathsf{Dom}(\Sigma)\end{array}}{\Sigma, a{:}K\ \mathsf{sig}}\ (\mathsf{S{\cdot}Kind})$$

$$\frac{\begin{array}{l}\Sigma\ \mathsf{sig}\\ \vdash_\Sigma \sigma{:}\mathsf{Type} \quad c \notin \mathsf{Dom}(\Sigma)\end{array}}{\Sigma, c{:}\sigma\ \mathsf{sig}}\ (\mathsf{S{\cdot}Type})$$

Context rules

$$\frac{\Sigma\ \mathsf{sig}}{\vdash_\Sigma \emptyset}\ (\mathsf{C{\cdot}Empty})$$

$$\frac{\begin{array}{l}\vdash_\Sigma \Gamma\\ \Gamma \vdash_\Sigma \sigma{:}\mathsf{Type} \ \ x \notin \mathsf{Dom}(\Gamma)\end{array}}{\vdash_\Sigma \Gamma, x{:}\sigma}\ (\mathsf{C{\cdot}Type})$$

Kind rules

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \mathsf{Type}}\ (\mathsf{K{\cdot}Type})$$

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

Family rules

$$\frac{\vdash_\Sigma \Gamma \quad a{:}K \in \Sigma}{\Gamma \vdash_\Sigma a : K}\ (\mathsf{F{\cdot}Const})$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.\tau : \mathsf{Type}}\ (\mathsf{F{\cdot}Pi})$$

$$\frac{\Gamma \vdash_\Sigma \sigma : \Pi x{:}\tau.K \quad \Gamma \vdash_\Sigma N : \tau}{\Gamma \vdash_\Sigma \sigma\, N : K[N/x]}\ (\mathsf{F{\cdot}App})$$

$$\frac{\Gamma \vdash_\Sigma \rho : \mathsf{Type} \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}}\ (\mathsf{F{\cdot}Lock})$$

$$\frac{\Gamma \vdash_\Sigma \sigma : K \quad \Gamma \vdash_\Sigma K' \quad K{=}_{\beta\mathcal{L}}K'}{\Gamma \vdash_\Sigma \sigma : K'}\ (\mathsf{F{\cdot}Conv})$$

Object rules

$$\frac{\vdash_\Sigma \Gamma \quad c{:}\sigma \in \Sigma}{\Gamma \vdash_\Sigma c : \sigma}\ (\mathsf{O{\cdot}Const})$$

$$\frac{\vdash_\Sigma \Gamma \quad x{:}\sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma}\ (\mathsf{O{\cdot}Var})$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma M : \tau}{\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau}\ (\mathsf{O{\cdot}Abs})$$

$$\frac{\Gamma \vdash_\Sigma M : \Pi x{:}\sigma.\tau \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma M\, N : \tau[N/x]}\ (\mathsf{O{\cdot}App})$$

$$\frac{\Gamma \vdash_\Sigma M : \rho \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]}\ (\mathsf{O{\cdot}Lock})$$

$$\frac{\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]\Gamma \vdash_\Sigma N : \sigma \quad \mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)}{\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho}\ (\mathsf{O{\cdot}Unlock})$$

$$\frac{\Gamma \vdash_\Sigma M : \sigma \quad \Gamma \vdash_\Sigma \tau : \mathsf{Type} \quad \sigma{=}_{\beta\mathcal{L}}\tau}{\Gamma \vdash_\Sigma M : \tau}\ (\mathsf{O{\cdot}Conv})$$

Figure 2. The $\mathsf{LF}_\mathcal{P}$ Type System

$$\frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\Pi x{:}\sigma.K \to_{\beta\mathcal{L}} \Pi x{:}\sigma'.K}\ (\mathsf{K{\cdot}\Pi_1{\cdot}}\beta\mathcal{L}) \qquad \frac{K \to_{\beta\mathcal{L}} K'}{\Pi x{:}\sigma.K \to_{\beta\mathcal{L}} \Pi x{:}\sigma.K'}\ (\mathsf{K{\cdot}\Pi_2{\cdot}}\beta\mathcal{L})$$

Figure 4. $\beta\mathcal{L}$-context-closure on kinds

$$\frac{K \to_{\beta\mathcal{L}} K'}{K{=}_{\beta\mathcal{L}}K'}\ (\mathsf{K{\cdot}Main{\cdot}Eq}_{\beta\mathcal{L}}) \qquad \frac{K{=}_{\beta\mathcal{L}}K'}{K'{=}_{\beta\mathcal{L}}K}\ (\mathsf{K{\cdot}Sym{\cdot}Eq}_{\beta\mathcal{L}})$$

$$\frac{}{K{=}_{\beta\mathcal{L}}K}\ (\mathsf{K{\cdot}Refl{\cdot}Eq}_{\beta\mathcal{L}}) \qquad \frac{K{=}_{\beta\mathcal{L}}K' \quad K'{=}_{\beta\mathcal{L}}K''}{K{=}_{\beta\mathcal{L}}K''}\ (\mathsf{K{\cdot}Trans{\cdot}Eq}_{\beta\mathcal{L}})$$

Figure 5. $\beta\mathcal{L}$-equivalence on kinds

$$\frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\Pi x{:}\sigma.\tau \to_{\beta\mathcal{L}} \Pi x{:}\sigma'.\tau}\,(\mathsf{F{\cdot}\Pi_1{\cdot}}\beta\mathcal{L}) \qquad \frac{\tau \to_{\beta\mathcal{L}} \tau'}{\Pi x{:}\sigma.\tau \to_{\beta\mathcal{L}} \Pi x{:}\sigma.\tau'}\,(\mathsf{F{\cdot}\Pi_2{\cdot}}\beta\mathcal{L})$$

$$\frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\sigma\, N \to_{\beta\mathcal{L}} \sigma'\, N}\,(\mathsf{F{\cdot}App_1{\cdot}}\beta\mathcal{L}) \qquad \frac{N \to_{\beta\mathcal{L}} N'}{\sigma\, N \to_{\beta\mathcal{L}} \sigma\, N'}\,(\mathsf{F{\cdot}App_2{\cdot}}\beta\mathcal{L})$$

$$\frac{N \to_{\beta\mathcal{L}} N'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N',\sigma}[\rho]}\,(\mathsf{F{\cdot}Lock_1{\cdot}}\beta\mathcal{L}) \qquad \frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho]}\,(\mathsf{F{\cdot}Lock_2{\cdot}}\beta\mathcal{L})$$

$$\frac{\rho \to_{\beta\mathcal{L}} \rho'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho']}\,(\mathsf{F{\cdot}Lock_3{\cdot}}\beta\mathcal{L})$$

Figure 6. $\beta\mathcal{L}$-context-closure on families

$$\frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\sigma =_{\beta\mathcal{L}} \sigma'}\,(\mathsf{F{\cdot}Main{\cdot}Eq}_{\beta\mathcal{L}}) \qquad \frac{\sigma =_{\beta\mathcal{L}} \sigma'}{\sigma' =_{\beta\mathcal{L}} \sigma}\,(\mathsf{F{\cdot}Sym{\cdot}Eq}_{\beta\mathcal{L}})$$

$$\frac{}{\sigma =_{\beta\mathcal{L}} \sigma}\,(\mathsf{F{\cdot}Refl{\cdot}Eq}_{\beta\mathcal{L}}) \qquad \frac{\sigma =_{\beta\mathcal{L}} \sigma' : K \quad \sigma' =_{\beta\mathcal{L}} \sigma''}{\sigma =_{\beta\mathcal{L}} \sigma''}\,(\mathsf{F{\cdot}Trans{\cdot}Eq}_{\beta\mathcal{L}})$$

Figure 7. $\beta\mathcal{L}$-equivalence on families

$$\frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\lambda x{:}\sigma.M \to_{\beta\mathcal{L}} \lambda x{:}\sigma'.M}\,(\mathsf{O{\cdot}}\lambda_1\mathsf{{\cdot}}\beta\mathcal{L}) \qquad \frac{M \to_{\beta\mathcal{L}} M'}{\lambda x{:}\sigma.M \to_{\beta\mathcal{L}} \lambda x{:}\sigma.M'}\,(\mathsf{O{\cdot}}\lambda_2\mathsf{{\cdot}}\beta\mathcal{L})$$

$$\frac{M \to_{\beta\mathcal{L}} M'}{M\, N \to_{\beta\mathcal{L}} M'\, N}\,(\mathsf{O{\cdot}App_1{\cdot}}\beta\mathcal{L}) \qquad \frac{N \to_{\beta\mathcal{L}} N'}{M\, N \to_{\beta\mathcal{L}} M\, N'}\,(\mathsf{O{\cdot}App_2{\cdot}}\beta\mathcal{L})$$

$$\frac{N \to_{\beta\mathcal{L}} N'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N',\sigma}[M]}\,(\mathsf{O{\cdot}Lock_1{\cdot}}\beta\mathcal{L}) \qquad \frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[M]}\,(\mathsf{O{\cdot}Lock_2{\cdot}}\beta\mathcal{L})$$

$$\frac{M \to_{\beta\mathcal{L}} M'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M']}\,(\mathsf{O{\cdot}Lock_3{\cdot}}\beta\mathcal{L}) \qquad \frac{N \to_{\beta\mathcal{L}} N'}{\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N',\sigma}[M]}\,(\mathsf{O{\cdot}Unlock_1{\cdot}}\beta\mathcal{L})$$

$$\frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N,\sigma'}[M]}\,(\mathsf{O{\cdot}Unlock_2{\cdot}}\beta\mathcal{L}) \qquad \frac{M \to_{\beta\mathcal{L}} M'}{\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M']}\,(\mathsf{O{\cdot}Unlock_3{\cdot}}\beta\mathcal{L})$$

Figure 8. $\beta\mathcal{L}$-context-closure on objects

$$\frac{M \to_{\beta\mathcal{L}} M'}{M =_{\beta\mathcal{L}} M'}\,(\mathsf{O{\cdot}Main{\cdot}Eq}_{\beta\mathcal{L}}) \qquad \frac{M =_{\beta\mathcal{L}} M'}{M' =_{\beta\mathcal{L}} M}\,(\mathsf{O{\cdot}Sym{\cdot}Eq}_{\beta\mathcal{L}})$$

$$\frac{}{M =_{\beta\mathcal{L}} M}\,(\mathsf{O{\cdot}Refl{\cdot}Eq}_{\beta\mathcal{L}}) \qquad \frac{M =_{\beta\mathcal{L}} M' \quad M' =_{\beta\mathcal{L}} M''}{M =_{\beta\mathcal{L}} M''}\,(\mathsf{O{\cdot}Trans{\cdot}Eq}_{\beta\mathcal{L}})$$

Figure 9. $\beta\mathcal{L}$-equivalence on objects context, the structure of the derivation itself or other structures and mechanisms not available at the object level. This is the case for substructural and program logics [1, 2, 8].

We have isolated a *library* of predicates on proof terms, whose patterns frequently occur in the examples. There are two main archetypes: the first states that a constant k occurs (with some modality $\mathcal{D}$) in subterms satisfying the decidable property $\mathcal{C}$, while the second states that free variables only occur (with some modality $\mathcal{D}$) in subterms satisfying the decidable property $\mathcal{C}$. By $\mathcal{D}$ we mean phrases such as: at least once, only once, as the rightmost, does not occur, etc. $\mathcal{C}$ can refer to the syntactic form of the subterm or to that of its type, the latter being the main reason for allowing predicates in $\mathsf{LF}_{\mathcal{P}}$ to access the context. As a side remark, we notice that often the constraints on the type of a subterm can be expressed as constraints on the subterm itself by simply introducing suitable type coercion constants. In [15], we present a basic library of auxiliary functions, which can be used to introduce external predicates of the above archetypes.

We start with the encoding of the well known case of untyped λ-calculus, with a call-by-value evaluation strategy. This allows us to illustrate also how to deal with free and bound variables. Then we discuss modal logics and we give a sketch of how to encode the non-commutative linear logic introduced in [21]. Another example, on program logics *à la* Hoare, appears in [15].

We state adequacy theorems, whose proofs appear in Appendix A.

For the sake of simplicity, in the following examples, we use the notations $\sigma \to \tau$ for $\Pi x{:}\sigma.\tau$ if $x \notin \mathsf{Fv}(\tau)$, and σ^{n+1} for the n-ary abstraction $\sigma \to \ldots \to \sigma$. Moreover, we will omit the type σ in $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]$, where it is clear from the context.

In the adequacy theorems, we will use the notion of *judgement in η-long normal form*, defined as follows:

Definition 2 (Judgements in η-long normal form)**.**
- An occurrence ξ of a constant or a variable in a term of a $\mathsf{LF}_{\mathcal{P}}$ judgement is fully applied and unlocked *with respect to its type or kind $\Pi\vec{x}_1{:}\vec{\sigma}_1.\vec{\mathcal{L}}_1[\ldots\Pi\vec{x}_n{:}\vec{\sigma}_n.\vec{\mathcal{L}}_n[\alpha]\ldots]$, where $\vec{\mathcal{L}}_1,\ldots,\vec{\mathcal{L}}_n$ are vectors of locks, if ξ appears in contexts of the form $\vec{\mathcal{U}}_n[(\ldots(\vec{\mathcal{U}}_1[\xi\vec{M}_1])\ldots)\vec{M}_n]$, where $\vec{M}_1,\ldots,\vec{M}_n$, $\vec{\mathcal{U}}_1,\ldots,\vec{\mathcal{U}}_n$ have the same arities of the corresponding vectors of Π's and locks.*
*- A term T in a judgement is in η-*lnf *if T is in normal form and every constant and variable occurrence in T is fully applied and unlocked w.r.t. its classifier in the judgement.*
*- A judgement is in η-*lnf *if all terms appearing in it are in η-lnf.*

3.1 The untyped λ-calculus

3.1.1 Free and bound variables.

Consider the well-known untyped λ-calculus:

$$M, N, \ldots ::= x \mid M\ N \mid \lambda x.M\ ,$$

with variables, application and abstraction. We model free variables of the object language as constants in $\mathsf{LF}_{\mathcal{P}}$, while retaining the full Higher-Order-Abstract-Syntax (HOAS) approach for modeling bindable and bound variables with variables of the metalanguage, thus delegating to the latter α-conversion and capture-avoiding substitution. Such an approach allows us to abide by the "closure under substitution" condition for external predicates, while retaining the ability to handle "open" terms.

The abovementioned "bindable" variables must not be confused neither with bound variables nor with free variables. For instance, the λ-term x (where the variable is free) will be encoded by means of the term $\vdash_\Sigma$ `(free n):term` for a suitable (encoding of) natural number `n` (see Definition 3 below). On the other hand the λ-term $\lambda x.x$ (where the variable is obviously bound) will be encoded by $\vdash_\Sigma$ λ`x:term.x`. However, when we "open" the abstraction $\lambda x.M$, considering the body M, we will encode the latter as `x:term` $\vdash_\Sigma$ $\epsilon_{\{x\}}$(`M`), where $\epsilon_{\{x\}}$ is the encoding function defined later in this section. In this case x is a *bindable* variable.

Definition 3 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ_λ for untyped λ-calculus)**.**

$$\begin{array}{lll} \texttt{nat, term} : \texttt{Type} & \texttt{0} : \texttt{nat} & \texttt{S} : \texttt{nat}^2 \\ \texttt{free} : \texttt{nat -> term} & \texttt{app} : \texttt{term}^3 & \texttt{lambda} : \texttt{term}^2\ \texttt{-> term} \end{array}$$

We use the natural numbers as standard abbreviations for repeated applications of `S` to `0`. Given an enumeration $\{x_i\}_{i\in\mathbb{N}\setminus\{0\}}$ of the variables in the untyped λ-calculus, we put:

$$\begin{array}{rcl} \epsilon_{\mathcal{X}}(x_i) & = & \left\{ \begin{array}{l} \texttt{xi}\qquad\ \text{, if } x_i \in \mathcal{X} \\ \texttt{free(i)}\text{, if } x_i \notin \mathcal{X} \end{array} \right. \\ \epsilon_{\mathcal{X}}(MN) & = & (\texttt{app}\ \epsilon_{\mathcal{X}}(M)\,\epsilon_{\mathcal{X}}(N)) \\ \epsilon_{\mathcal{X}}(\lambda x.M) & = & (\texttt{lambda}\ \lambda\texttt{x:term.}\epsilon_{\mathcal{X}\cup\{\texttt{x}\}}(M)) \end{array}$$

where in the latter clause, $x \notin \mathcal{X}$.

Theorem 2 (Adequacy of syntax)**.** *Given an enumeration $\{x_i\}_{i\in\mathbb{N}\setminus\{0\}}$ of the variables in the λ-calculus, the encoding function $\epsilon_{\mathcal{X}}$ is a bijection between the λ-calculus terms with bindable variables in $\mathcal{X}$ and the terms M derivable in judgements $\Gamma \vdash_{\Sigma_\lambda} M$:* `term` *in η-lnf, where $\Gamma = \{\texttt{x} : \texttt{term} \mid x \in \mathcal{X}\}$. Moreover, the encoding is compositional, i.e. for a term M, with bindable variables in $\mathcal{X} = \{x_1,\ldots,x_k\}$, and $N_1,\ldots,N_k$, with bindable variables in $\mathcal{Y}$, the following holds: $\epsilon_{\mathcal{X}}(M[N_1,\ldots,N_k/x_1,\ldots,x_k]) = \epsilon_{\mathcal{X}}(M)[\epsilon_{\mathcal{Y}}(N_1),\ldots,\epsilon_{\mathcal{Y}}(N_k)/x_1,\ldots,x_k]$.*

Proof. See Appendix A.2.1. □

3.1.2 Untyped λ-calculus and call-by-value reduction strategy.

The call-by-value (CBV) evaluation strategy can be specified by:

$$\frac{}{\vdash_{CBV} M = M}\,(\mathsf{refl}) \qquad \frac{\vdash_{CBV} N = M}{\vdash_{CBV} M = N}\,(\mathsf{symm})$$

$$\frac{\vdash_{CBV} M = N \quad \vdash_{CBV} N = P}{\vdash_{CBV} M = P}\,(\mathsf{trans})$$

$$\frac{\vdash_{CBV} M = N \quad \vdash_{CBV} M' = N'}{\vdash_{CBV} MM' = NN'}\,(\mathsf{app})$$

$$\frac{v \text{ is a value}}{\vdash_{CBV} (\lambda x.M)v = M[v/x]}\,(\beta_{\mathsf{v}}) \qquad \frac{\vdash_{CBV} M = N}{\vdash_{CBV} \lambda x.M = \lambda x.N}\,(\xi_{\mathsf{v}})$$

Definition 4 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ_{CBV} for λ-calculus CBV reduction)**.** *We extend the signature of Definition 3 as follows:*

$$\begin{array}{lll} \texttt{triple} & : & \texttt{Type} \\ \langle\_,\_,\_\rangle & : & \texttt{term -> term}^2\ \texttt{-> term}^2\ \texttt{-> triple} \\ \texttt{eq} & : & \texttt{term -> term -> Type} \\ \texttt{refl} & : & \Pi\texttt{M:term.(eq M M)} \\ \texttt{symm} & : & \Pi\texttt{M:term.}\Pi\texttt{N:term.(eq N M) -> (eq M N)} \\ \texttt{trans} & : & \Pi\texttt{M:term.}\Pi\texttt{N:term.}\Pi\texttt{P:term.} \\ & & \qquad\qquad\texttt{(eq M N) -> (eq N P) -> (eq M P)} \\ \texttt{eq\_app} & : & \Pi\texttt{M,N,M',N':term.} \\ \multicolumn{3}{l}{\quad\texttt{(eq M N) -> (eq M' N') -> (eq (app M M') (app N N'))}} \\ \texttt{betav} & : & \Pi\texttt{M:term}^2\texttt{.}\Pi\texttt{N:term.}\ \mathcal{L}^{\mathit{Val}}_{\texttt{N}}\texttt{[(eq (app (lambda M) N) (M N))]} \\ \texttt{csiv} & : & \Pi\texttt{M,N:term}^2\texttt{.}\Pi\texttt{x:term.} \\ \multicolumn{3}{l}{\ \mathcal{L}^{\xi}_{\langle\texttt{x,M,N}\rangle,\texttt{triple}}\texttt{[(eq (M x)(N x))->(eq (lambda M)(lambda N))]}} \end{array}$$

where the predicates Val*, ξ are defined as follows and* `triple` *is the obvious type of triples of terms with types* `term`*,* $\texttt{term}^2$ *and* $\texttt{term}^2$*:*
- Val $(\Gamma \vdash_\Sigma \texttt{N:term})$ *holds iff either* `N` *is an abstraction or* `N` *is a constant (i.e. a term of the shape* `(free i)`*);*

- $\xi(\Gamma \vdash_\Sigma \langle \texttt{x,M,N}\rangle \texttt{:triple})$ *holds iff* `x` *is a constant (i.e. a term of the shape* `(free i)`*),* `M` *and* `N` *are closed and* `x` *does not occur in* `M` *and* `N`.

Theorem 3 (Adequacy of CBV reduction)**.** *Given an enumeration $\{x_i\}_{i \in \mathbb{N} \setminus \{0\}}$ of the variables in the λ-calculus, there is a bijection between derivations of the judgment $\vdash_{CBV} M = N$ on terms with no bindable variables in the CBV λ-calculus and proof terms* `h` *such that $\vdash_{\Sigma_{CBV}} \texttt{h} : (\texttt{eq}\, \epsilon_\emptyset(M)\, \epsilon_\emptyset(N))$ is in η-long normal form.*

Proof. See Appendix A.2.2. □

3.2 Substructural logics

In many formal systems, rules are subject to side conditions and structural constraints on the shape of assumptions or premises. Typical examples are the necessitation rule or the □-introduction rules in Modal logics (see, e.g., [1, 2, 8]).

For the sake of readability, in the following we will often use an *infix* notation for encoding binary logic operators.

3.2.1 Modal Logics in Hilbert style.

In this example, we show how $\mathsf{LF}_\mathcal{P}$ allows to encode smoothly logical systems with "rules of proof" as well as "rules of derivation". The former apply only to premises which do not depend on any assumption, such as *necessitation*, while the latter are the usual rules which apply to all premises, such as *modus ponens*. The idea is to use suitable "lock types" in rules of proof and "standard" types in the rules of derivation.

$$\begin{array}{lcl}
\mathsf{A_1} & : & \phi \to (\psi \to \phi) \\
\mathsf{A_2} & : & (\phi \to (\psi \to \xi)) \to (\phi \to \psi) \to (\phi \to \xi) \\
\mathsf{A_3} & : & (\neg\phi \to \neg\psi) \to ((\neg\phi \to \psi) \to \phi) \\
\mathsf{K} & : & \Box(\phi \to \psi) \to (\Box\phi \to \Box\psi) \\
\mathsf{T} & : & \Box\phi \to \phi \\
\mathsf{4} & : & \Box\phi \to \Box\Box\phi \\
\mathsf{MP} & : & \dfrac{\phi \quad \phi \to \psi}{\psi} \\
\mathsf{NEC} & : & \dfrac{\phi}{\Box\phi}
\end{array}$$

Figure 10. Hilbert style rules for Modal Logic S_4

By way of example, we give the signature for classical S_4 (see Figure 11) in Hilbert style (see Figure 10), which features necessitation (rule NEC in Figure 10) as a rule of proof. Due to lack of space, we limit the encoding in Figure 11 to the most significant cases. We make use of the predicate $\mathit{Closed}(\Gamma \vdash_\Sigma \texttt{m}{:}\mathit{True}(\phi))$, which holds iff "all free variables occurring in `m` belong to a subterm which is typable with `o`". This is precisely what is needed to correctly encode the notion of rule of proof, if `o` is the type of propositions. Indeed, if all the free variables of a proof term satisfy such a condition, it is clear, by inspection of the η-lnfs, that there cannot be free variables of type $\mathit{True}(\dots)$ in the proof term, *i.e.* the encoded modal formula does not depend on any assumption[1] (see [15] for a formal specification of the predicate). This example requires that predicates inspect the environment and be defined on *typed judgements*, as indeed is the case in $\mathsf{LF}_\mathcal{P}$. The above predicate is well-behaved. As in the previous examples, we ensure a sound derivation in $\mathsf{LF}_\mathcal{P}$ of a proof of $\Box\phi$, by locking the type $\texttt{True}(\Box\phi)$ in the conclusion of `NEC` (see Figure 11).

Adequacy theorems are rather trivial to state and prove; as usual we define an encoding function $\epsilon_\mathcal{X}$ on formulæ with free variables in $\mathcal{X}$ as follows, representing atomic formulæ by means of $\mathsf{LF}_\mathcal{P}$ metavariables:

- $\epsilon_\mathcal{X}(x) = \texttt{x}$, where $x \in \mathcal{X}$;
- $\epsilon_\mathcal{X}(\phi \to \psi) = \epsilon_\mathcal{X}(\phi) \to \epsilon_\mathcal{X}(\psi)$;
- $\epsilon_\mathcal{X}(\neg\phi) = \neg\epsilon_\mathcal{X}(\phi)$;
- $\epsilon_\mathcal{X}(\Box\phi) = \Box\epsilon_\mathcal{X}(\phi)$.

Then, we can prove by structural induction on formulæ, the following theorem:

Theorem 4 (Adequacy of S_4 formulæ syntax)**.** *The encoding function $\epsilon_\mathcal{X}$ is a bijection between the modal logic formulæ with free variables in $\mathcal{X}$ and the terms ϕ derivable in judgements $\Gamma \vdash_{\Sigma_\Box} \phi : \texttt{o}$ in η-lnf, where $\Gamma = \{\texttt{x} : \texttt{o} \mid x \in \mathcal{X}\}$. Moreover, the encoding is compositional, i.e. for a formula ϕ, with free variables in $\mathcal{X} = \{x_1, \dots, x_k\}$, and $\psi_1, \dots, \psi_k$, with free variables in $\mathcal{Y}$, the following holds: $\epsilon_\mathcal{X}(\phi[\psi_1, \dots, \psi_k/x_1, \dots, x_k]) = \epsilon_\mathcal{X}(\phi)[\epsilon_\mathcal{Y}(\psi_1), \dots, \epsilon_\mathcal{Y}(\psi_k)/x_1, \dots, x_k]$.*

If we denote by $\phi_1, \dots, \phi_n \vdash \phi$ the derivation of the truth of a formula ϕ, depending on the assumptions $\phi_1, \dots, \phi_n$, in the Hilbert-style modal logic S_4, the adequacy of our encoding can then be stated by the following theorem:

Theorem 5 (Adequacy of S_4 truth system in Hilbert-style)**.** *There is a bijection between derivations $\phi_1, \dots, \phi_k \vdash \phi$ in the Hilbert-style S_4 modal logic and proof terms* `h` *such that $\Gamma \vdash_\Sigma \texttt{h} : (\texttt{True}\ \epsilon_\mathcal{X}(\phi_1 \to \dots \to \phi_k \to \phi))$ in η-long normal form, where $\mathcal{X} = \{x_1, \dots, x_n\}$ is the set of propositional variables occurring in $\phi_1, \dots, \phi_k, \phi$ and $\Gamma = \{\texttt{x1} : \texttt{o}, \dots, \texttt{xn} : \texttt{o}\}$.*

$$\begin{array}{lll}
\texttt{o : Type} \quad \to : \texttt{o}^3 \quad \neg : \texttt{o}^2 \quad \Box : \texttt{o}^2 & & \texttt{True : o -> Type} \\
\texttt{A1} & : \Pi\phi,\psi{:}\texttt{o}. & \texttt{True}(\phi{\to}(\psi{\to}\phi)) \\
\texttt{K} & : \Pi\phi,\psi{:}\texttt{o}. & \texttt{True}(\Box(\phi{\to}\psi){\to}(\Box\phi{\to}\Box\psi)) \\
\texttt{MP} & : \Pi\phi{:}\texttt{o}.\Pi\psi{:}\texttt{o}. & \texttt{True}(\phi)\ \texttt{->}\ \texttt{True}(\phi{\to}\psi)\ \texttt{->}\ \texttt{True}(\psi) \\
\texttt{NEC} & : \Pi\phi{:}\texttt{o}.\Pi\texttt{m}{:}\texttt{True}(\phi). & \mathcal{L}_{\texttt{m}}^{\mathit{Closed}}[\texttt{True}(\Box\phi)]
\end{array}$$

Figure 11. The signature Σ for classic S_4 Modal Logic in Hilbert style

3.2.2 Modal Logics S_4 and S_5 in Prawitz style.

In $\mathsf{LF}_\mathcal{P}$, one can also accommodate other modal logics, such as classical Modal Logics S_4 and S_5 in Natural Deduction style, as defined by Prawitz, which have rules with rather elaborate restrictions on the shape of subformulae where assumptions occur. Figure 12 shows some of the rules common to both systems and all specific rules of S_4 and S_5. In order to illustrate the flexibility of the system, the rule for S_4 is given in the form which allows cut-elimination. Figure 13 shows their encoding in $\mathsf{LF}_\mathcal{P}$. Again, the crucial role is played by a predicate, namely, $\mathit{Boxed}(\,)$. The intended meaning is that $\mathit{Boxed}(\Gamma \vdash_\Sigma \texttt{m}{:}\ \texttt{True}(\phi))$ holds in the case of S_4 iff the occurrences of free variables of `m` occur in subterms whose type has the shape $\texttt{True}(\Box\psi)$ or is `o`. In the case of S_5 the predicate holds iff the variables of `m` have type $\texttt{True}(\Box\psi)$ or $\texttt{True}(\neg\Box\psi)$ or occur in subterms whose type is `o`. It is easy to check that these predicates are well behaved. Again, the "trick" to ensure a sound derivation in $\mathsf{LF}_\mathcal{P}$ of a proof of $\Box\phi$ is to lock appropriately the type $\texttt{True}(\Box\phi)$ in the conclusion of the introduction rule `BoxI` (see Figure 13).

The problem of representing, in a sound way, modal logics in logical frameworks based on type theory is well-known in the literature [1, 2, 8]. In our approach, we avoid the explicit introduction in the encodings of extra-judgments and structures, as in [1, 2, 8],

[1] Another way of specifying such a property is to require that "all free variables occurring in `m` have a simple type over *o*".

$$\frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}\,(\wedge\mathsf{I}) \qquad \frac{\Gamma \vdash \phi \vee \psi \quad \Gamma, \phi \vdash \xi \quad \Gamma, \psi \vdash \xi}{\Gamma \vdash \xi}\,(\vee\mathsf{E})$$

$$\frac{\Delta \vdash \Box\Gamma \quad \Box\Gamma \vdash \phi}{\Delta \vdash \Box\phi}\,(\Box\mathsf{I}\cdot\mathsf{S}_4) \qquad \frac{\Box\Gamma_0, \neg\Box\Gamma_1 \vdash \phi}{\Box\Gamma_0, \neg\Box\Gamma_1 \vdash \Box\phi}\,(\Box\mathsf{I}\cdot\mathsf{S}_5)$$

$$\frac{\Gamma \vdash \Box\phi}{\Gamma \vdash \phi}\,(\Box\mathsf{E}\cdot\mathsf{S}_4) \qquad \frac{\Gamma \vdash \Box\phi}{\Gamma \vdash \phi}\,(\Box\mathsf{E}\cdot\mathsf{S}_5) \qquad \frac{\Gamma, \neg\phi \vdash \phi}{\Gamma \vdash \phi}\,(\mathsf{RAA})$$

Figure 12. Some Modal Logic rules (common and $S_{4,5}$ rules) in Natural Deduction style

$$\texttt{o:Type} \qquad \texttt{and:o}^3 \qquad \texttt{or:o}^3 \qquad \rightarrow\texttt{:o}^3 \qquad \neg\texttt{:o}^2 \qquad \Box\texttt{:o}^2$$

$$\begin{array}{lll}
\texttt{True} & : & \texttt{o -> Type} \\
\texttt{AndI} & : & \Pi\phi\texttt{,}\psi\texttt{:o. True(}\phi\texttt{) -> True(}\psi\texttt{) -> True(}\phi\texttt{ and }\psi\texttt{)} \\
\texttt{OrE} & : & \Pi\phi\texttt{,}\psi\texttt{,}\xi\texttt{:o.True(}\phi\texttt{ or }\psi\texttt{)->(True(}\phi\texttt{)->True(}\xi\texttt{))} \\
 & & \qquad\qquad\texttt{->(True(}\psi\texttt{)->True(}\xi\texttt{))->True(}\xi\texttt{)} \\
\texttt{RAA} & : & \Pi\phi\texttt{:o. (True(}\neg\phi\texttt{) -> True(}\phi\texttt{)) -> True(}\phi\texttt{)} \\
\texttt{BoxI} & : & \Pi\phi\texttt{:o. }\Pi\texttt{m:True(}\phi\texttt{). }\mathcal{L}^{\mathit{Boxed}}_{\texttt{m}}[\texttt{True(}\Box\phi\texttt{)}] \\
\texttt{BoxE} & : & \Pi\phi\texttt{:o. }\Pi\texttt{m:True(}\Box\phi\texttt{). True(}\phi\texttt{)}
\end{array}$$

Figure 13. The signature Σ_S for classic S_4 Modal Logic in $\mathsf{LF}_{\mathcal{P}}$

by delegating such machinery to an *external oracle* by means of locks.

For what concerns the adequacy of our encoding, we can state Theorems 6 and 7 below. As in the previous case study, we first define an encoding function $\epsilon_{\mathcal{X}}$ on formulæ with free variables in $\mathcal{X}$ as follows, representing atomic formulæ by means of $\mathsf{LF}_{\mathcal{P}}$ metavariables:

- $\epsilon_{\mathcal{X}}(x) = \texttt{x}$, where $x \in \mathcal{X}$;
- $\epsilon_{\mathcal{X}}(\phi \wedge \psi) = \epsilon_{\mathcal{X}}(\phi)$ $\texttt{and}$ $\epsilon_{\mathcal{X}}(\psi)$;
- $\epsilon_{\mathcal{X}}(\phi \vee \psi) = \epsilon_{\mathcal{X}}(\phi)$ $\texttt{or}$ $\epsilon_{\mathcal{X}}(\psi)$;
- $\epsilon_{\mathcal{X}}(\phi \rightarrow \psi) = \epsilon_{\mathcal{X}}(\phi) \rightarrow \epsilon_{\mathcal{X}}(\psi)$;
- $\epsilon_{\mathcal{X}}(\neg\phi) = \neg\epsilon_{\mathcal{X}}(\phi)$;
- $\epsilon_{\mathcal{X}}(\Box\phi) = \Box\epsilon_{\mathcal{X}}(\phi)$.

Then, we can prove by structural induction on formulæ, the following theorem:

Theorem 6 (Adequacy of S_4/S_5 formulæ syntax)**.** *The encoding function $\epsilon_{\mathcal{X}}$ is a bijection between the modal logic formulæ with free variables in $\mathcal{X}$ and the terms ϕ derivable in judgements $\Gamma \vdash_{\Sigma_\Box} \phi : \texttt{o}$ in η-lnf, where $\Gamma = \{\texttt{x} : \texttt{o} \mid x \in \mathcal{X}\}$. Moreover, the encoding is compositional, i.e. for a formula ϕ, with free variables in $\mathcal{X} = \{x_1, \ldots, x_k\}$, and $\psi_1, \ldots, \psi_k$, with free variables in $\mathcal{Y}$, the following holds: $\epsilon_{\mathcal{X}}(\phi[\psi_1, \ldots, \psi_k/x_1, \ldots, x_k]) = \epsilon_{\mathcal{X}}(\phi)[\epsilon_{\mathcal{Y}}(\psi_1), \ldots, \epsilon_{\mathcal{Y}}(\psi_k)/x_1, \ldots, x_k]$.*

The adequacy of the truth system of S_4/S_5 modal logic can then be proved by structural induction on derivations of the judgment $\Gamma \vdash \phi$:

Theorem 7 (Adequacy of modal logic S_4/S_5)**.** *Given a set of propositional variables $\mathcal{X} = \{x_1, \ldots, x_n\}$ occurring in formulæ $\phi_1, \ldots, \phi_k, \phi$, there is a bijection between derivations of the judgment $\{\phi_1, \ldots, \phi_k\} \vdash \phi$ in the S_4/S_5 modal logic and proof terms* $\texttt{h}$ *such that* $\Gamma \vdash_\Sigma \texttt{h} : (\texttt{True}\ \epsilon_{\mathcal{X}}(\phi))$ *in η-long normal form, where* $\Gamma = \{\texttt{x1} : \texttt{o}, \ldots, \texttt{xn} : \texttt{o}, \texttt{h1} : (\texttt{True}\ \epsilon_{\mathcal{X}}(\phi_1)), \ldots, \texttt{hk} : (\texttt{True}\ \epsilon_{\mathcal{X}}\ (\phi_k))\}$.

3.2.3 Non-commutative linear logic (NCLL).

In this section we outline an encoding in $\mathsf{LF}_{\mathcal{P}}$ of a substructural logic like the one presented in [21]. Take, for instance, the rules for the *ordered variables* and the $\twoheadrightarrow$ introduction/elimination rules:

$$\frac{}{\Gamma; \cdot; z{:}A \vdash z{:}A}\,(\mathsf{ovar}) \qquad \frac{\Gamma; \Delta; (\Omega, z{:}A) \vdash M{:}B}{\Gamma; \Delta; \Omega \vdash \lambda^{>} z{:}A.M{:}A \twoheadrightarrow B}\,(\twoheadrightarrow \mathsf{I})$$

$$\frac{\Gamma; \Delta_1; \Omega_1 \vdash M{:}A \twoheadrightarrow B \quad \Gamma; \Delta_2; \Omega_2 \vdash N{:}A}{\Gamma; (\Delta_1 \bowtie \Delta_2); (\Omega_1, \Omega_2) \vdash M^{>} N{:}B}\,(\twoheadrightarrow \mathsf{E})$$

In this system “ordered assumptions occur exactly once and in the order they were made”. In order to encode the condition about the occurrence of z as the last variable in the ordered context in the introduction rule, it is sufficient to make the observation that in an LF-based logical framework this information is fully recorded in the proof term. The last assumption made is the rightmost variable, the first is the leftmost. Therefore, we can, in $\mathsf{LF}_{\mathcal{P}}$, introduce suitable predicates in order to enforce such constraints, without resorting to complicated encodings. In the following, we present an embedding of this ordered fragment of NCLL into $\mathsf{LF}_{\mathcal{P}}$. Our encoding is a *shallow* one in the sense that we are not interested in representing explicitly the proof terms of the original system (see, e.g., [21]). Hence, we represent only types as formulæ, discarding terms. The encodings of rules $\twoheadrightarrow I$ and $\twoheadrightarrow E$ are:

$\texttt{impRightIntro:}\ \Pi\texttt{A,B:o.}\Pi\texttt{M:(True A)->(True B).}$
$\qquad\qquad\qquad \mathcal{L}^{\mathit{Rightmost}}_{\texttt{M,(True A)}->\texttt{(True B)}}[(\texttt{True (impRight A B)})]$,
and
$\texttt{impRightElim:}\ \Pi\texttt{A,B:o.(True (impRight A B))->(True A)->(True B)}$
[*** Alternative version:
$\texttt{impRightElim:}\Pi\texttt{A,B:o.}\Pi\texttt{M:(True (impRight A B)).}\Pi\texttt{N:(True A).}$
$\qquad\qquad\qquad\qquad\qquad \mathcal{L}^{\mathit{Sep}}_{\langle\texttt{M,N}\rangle,\texttt{(pair A B)}}[(\texttt{True B})]$,
where (pair A B) is the obious type of pairs of terms with types (True (impRight A B)) and (True A), respectively, and the predicate $\mathit{Sep}(\Gamma \vdash \langle\texttt{M, N}\rangle{:}(\texttt{pair A B}))$ holds iff M and N have no common variables of type (True C) for some C. ***]
where True:o->Type is the truth judgment on formulæ (represented by type o) and $\texttt{impRight:o}^3$ represents the $\twoheadrightarrow$ constructor of *right ordered implications*.

Finally, $\mathit{Rightmost}(\Gamma \vdash_\Sigma \texttt{M:(True A)->(True B)})$ is the predicate checking that M is an abstraction in normal form (i.e., $\texttt{M}\equiv \lambda\texttt{z} : (\texttt{True A}).\texttt{M'}$ with M' in normal form), and that the bound variable z occurs only once and as the rightmost free one in M'.

Notice that in the encoding of rule $\twoheadrightarrow_E$ we have not enforced any conditions on the free variables occurring in the involved terms. Indeed, the requirement that the ordered contexts Ω_1 and Ω_2 do not have variables in common will be stated by means of the following adequacy theorem:

Theorem 8 (Adequacy)**.** *Given a set of atomic formulæ $\mathcal{X} = \{P_1, \ldots, P_n\}$, occurring in formulæ $A_1, \ldots, A_k, A$, there is a bijection between derivations of the judgment $A_1, \ldots, A_k \vdash A$ in non-commutative linear logic and proof terms* $\texttt{h}$ *such that* $\Gamma_{\mathcal{X}}, \texttt{h1}{:}(\texttt{True}\ \epsilon_{\mathcal{X}}(A_1)), \ldots, \texttt{hk}{:}(\texttt{True}\ \epsilon_{\mathcal{X}}(A_k))$ $\vdash \texttt{h} : (\texttt{True}\ \epsilon_{\mathcal{X}}(A))$ *in η-long normal form, where the variables* $\texttt{h1}, \ldots, \texttt{hk}$ *occur in* $\texttt{h}$ *only once and in the order they are introduced in the derivation context and $\Gamma_{\mathcal{X}}$ is the context* P1:o,...,Pn:o *representing the object language propositional formulæ $P_1, \ldots, P_n$.*

Obviously, carrying out a deep embedding of the system, one could enforce the condition about the variables occurring in the ordered contexts by means of a suitable lock at the level of the proof terms (see, e.g., [15]).

As far as we know, this is the first example (see the discussion in, e.g., [8]) of an encoding of non-commutative linear logic in an LF-like framework.

4. Conclusions and Future Work

In this paper, we have presented an extension of the Edinburgh LF, which internalizes external oracles in the form of a $\diamond$ modal type constructor. Using $\mathsf{LF}_{\mathcal{P}}$, we have illustrated how we can factor out the complexity of encoding logical systems which are awkward in LF, *e.g.* Modal Logics and substructural logics, including non-commutative Linear Logic. More examples appear in [15], and others can be easily carried out, *e.g.* $\mathsf{LF}_{\mathcal{P}}$ within $\mathsf{LF}_{\mathcal{P}}$.

We believe that $\mathsf{LF}_{\mathcal{P}}$ provides a modular platform that can streamline the encoding of logics with arbitrary structural side-conditions in rules, *e.g.* involving, say, the number of applications of specific rules. We just need to extend the library of predicates.

In $\mathsf{LF}_{\mathcal{P}}$, one can easily incorporate systems which separate derivation and computation. *E.g.* the rule

$$\frac{A \to B \quad A \equiv C \quad C}{B}$$

in *Deduction Modulo* can be represented as:

$$\supseteq_{\equiv} : \Pi A, B, C{:}o.\Pi x{:}\mathit{True}(A \to B).\Pi y{:}\mathit{True}(C).\mathcal{L}^{\equiv}_{A,C}[\mathit{True}(B)]$$

We believe that our framework can also be very helpful in modeling dynamic and reactive systems: for example bio-inspired systems, where reactions of chemical processes take place only if some extra structural or temporal conditions hold, or process algebras. Often, in the latter systems, no assumptions can be made about messages exchanged through the communication channels. Indeed, it could be the case that a redex, depending on the result of a communication, can remain stuck until a "good" message arrives from a given channel, firing in that case an appropriate reduction (this is a common situation in many protocols, where "bad" requests are ignored and "good ones" are served). Such dynamic (run-time) behavior could hardly be captured by a rigid type discipline, where bad terms and hypotheses are ruled out *a priori* ([17]).

The machinery of lock derivations is akin to δ-rules *à la* Mitschke, see [3], when we take lock rules, at object level, as δ-rules releasing their argument when the condition is satisfied. This connection can be pursued further. For instance, we can use the untyped object language of $\mathsf{LF}_{\mathcal{P}}$ to support the "design by contract" programming paradigm. We illustrate this, using the predecessor function on natural numbers, which can be applied only to positive arguments. This control can be expressed using object level locks as $\lambda x{:}nat.\mathcal{L}^{x>0}_{x,nat}[x-1]$. More generally, if we want to enforce a pre-condition $\mathcal{P}$ on M and a post-condition $\mathcal{Q}$ on the result of the computation FM, we can easily express it in $\mathsf{LF}_{\mathcal{P}}$ by means of $\mathcal{L}^{\mathcal{P}}_{M}[\mathcal{L}^{\mathcal{Q}}_{(FM)}[(FM)]]$.

References

[1] A. Avron, F. Honsell, I. A. Mason, and R. Pollack. Using typed lambda calculus to implement formal systems on a machine. *Journal of Automated Reasoning*, 9:309–354, 1992. ISSN 0168-7433. URL `http://dx.doi.org/10.1007/BF00245294`.

[2] A. Avron, F. Honsell, M. Miculan, and C. Paravano. Encoding Modal Logics in Logical Frameworks. *Studia Logica*, 60(1):161–208, 1998.

[3] H. Barendregt. *Lambda Calculus: its Syntax and Semantics*. North Holland, 1984.

[4] H. Barendregt and E. Barendsen. Autarkic computations in formal proofs. *Journal of Automated Reasoning*, 28:321–336, 2002. ISSN 0168-7433. URL `http://dx.doi.org/10.1023/A:1015761529444`.

[5] G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. Pure Pattern Type Systems. In *POPL'03*, pages 250–261. The ACM Press, 2003.

[6] F. Blanqui, J.-P. Jouannaud, and P.-Y. Strub. From formal proofs to mathematical proofs: a safe, incremenental way for building in first-order decision procedures. In *IFIP TCS*, volume 273, pages 349–365, 2008.

[7] H. Cirstea, C. Kirchner, and L. Liquori. The Rho Cube. In *FOSSACS'01*, volume 2030 of *LNCS*, pages 166–180, 2001.

[8] K. Crary. Higher-order representation of substructural logics. In *ICFP '10*, pages 131–142. ACM, 2010. ISBN 978-1-60558-794-3. doi: http://doi.acm.org/10.1145/1863543.1863565. URL `http://doi.acm.org/10.1145/1863543.1863565`.

[9] G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31:33–72, 2003. ISSN 0168-7433.

[10] J.-C. Filliâtre and C. Marché. The Why/Krakatoa/Caduceus platform for deductive program verification. In *CAV'07*, volume 4590 of *LNCS*, 2007.

[11] R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *Journal of the ACM*, 40(1):143–184, 1993. Preliminary version in Proc. of LICS'87.

[12] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40:143–184, January 1993. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/138027.138060. URL `http://doi.acm.org/10.1145/138027.138060`.

[13] F. Honsell, M. Lenisa, and L. Liquori. A Framework for Defining Logical Frameworks. *Volume in Honor of G. Plotkin, ENTCS*, 172: 399–436, 2007.

[14] F. Honsell, M. Lenisa, L. Liquori, and I. Scagnetto. A conditional logical framework. In *LPAR'08*, volume 5330 of *LNCS*, pages 143–157, 2008.

[15] F. Honsell, M. Lenisa, L. Liquori, P. Maksimovic, and I. Scagnetto. LF_P – A Logical Framework with External Predicates. Technical report, Università di Udine, Italy, 2012.

[16] D. Licata and R. Harper. A universe of binding and computation. In *ICFP '09*, pages 123–134. ACM, 2009.

[17] A. Nanevski, F. Pfenning, and B. Pientka. Contextual Model Type Theory. *ACM Transactions on Computational Logic*, 9(3), 2008.

[18] F. Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *LICS'93*, pages 221–230, 1993.

[19] B. Pientka and J. Dunfield. Programming with proofs and explicit contexts. In *PPDP'08*, pages 163–173. ACM, 2008.

[20] B. Pientka and J. Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In J. Giesl and R. Hähnle, editors, *Automated Reasoning*, volume 6173 of *Lecture Notes in Computer Science*, pages 15–21. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-14202-4.

[21] J. Polakow and F. Pfenning. Natural deduction for intuitionistic non-commutative linear logic. In *TLCA'99*, volume 1581 of *LNCS*, pages 644–644, 1999.

[22] A. Poswolsky and C. Schürmann. System description: Delphin - a functional programming language for deductive systems. *Electr. Notes Theor. Comput. Sci.*, 228:113–120, 2009.

[23] A. Stump. Proof checking technology for satisfiability modulo theories. *Electronic Notes in Theoretical Computer Science*, 228(0):121 – 133, 2009. ISSN 1571-0661. Proc. of the International Workshop on Logical Frameworks and Metalanguages: Theory and Practice (LFMTP 2008).

A. Appendix

A.1 Properties of $\mathsf{LF}_{\mathcal{P}}$

Strong normalization.

In order to prove strong normalization of $\mathsf{LF}_{\mathcal{P}}$, we will rely on the strong normalization of LF, as proven in [12].

First, we will introduce the function $^{-\mathcal{UL}} : \mathsf{LF}_{\mathcal{P}} \to \mathsf{LF}$, which maps $\mathsf{LF}_{\mathcal{P}}$ terms into LF terms, by deleting the $\mathcal{L}$ and $\mathcal{U}$ symbols from an $\mathsf{LF}_{\mathcal{P}}$ term, while preserving all of the relevant information, in the following manner:

$$\begin{aligned}
([[\mathsf{Type}|a|c|x]])^{-\mathcal{UL}} &= [[Type|a|c|x]],\\
([[\Pi|\lambda]]x{:}\sigma.T)^{-\mathcal{UL}} &= [[\Pi|\lambda]]x{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}},\\
(T\,M)^{-\mathcal{UL}} &= T^{-\mathcal{UL}}\,M^{-\mathcal{UL}},\\
([[\mathcal{L}|\mathcal{U}]]^{\mathcal{P}}_{N,\sigma}[T])^{-\mathcal{UL}} &= (\lambda x_f{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}})N^{-\mathcal{UL}},
\end{aligned}$$

where, in the last item, x_f is a variable which does not have free occurrences in T. Here, it should be noticed that, although we have decided to remove abstractions in families in $\mathsf{LF}_{\mathcal{P}}$, we still, using $^{-\mathcal{UL}}$, translate $\mathsf{LF}_{\mathcal{P}}$-terms into full-fledged LF-terms, including those with abstractions in families. This is required so that the N and σ, which index the $\mathcal{L}$ and $\mathcal{U}$ symbols, are not lost. We can naturally extend $^{-\mathcal{UL}}$ to signatures and contexts of $\mathsf{LF}_{\mathcal{P}}$, obtaining signatures and contexts of LF:

$$\begin{aligned}
(\emptyset)^{-\mathcal{UL}} &= \emptyset,\\
(\Sigma, a{:}K)^{-\mathcal{UL}} &= \Sigma^{-\mathcal{UL}}, a^{-\mathcal{UL}}{:}K^{-\mathcal{UL}},\\
(\Sigma, c{:}\sigma)^{-\mathcal{UL}} &= \Sigma^{-\mathcal{UL}}, c^{-\mathcal{UL}}{:}\sigma^{-\mathcal{UL}},\\
(\emptyset)^{-\mathcal{UL}} &= \emptyset,\\
(\Gamma, x{:}\sigma)^{-\mathcal{UL}} &= \Gamma^{-\mathcal{UL}}, x^{-\mathcal{UL}}{:}\sigma^{-\mathcal{UL}}.
\end{aligned}$$

and then to judgements of $\mathsf{LF}_{\mathcal{P}}$, obtaining judgements of LF:

$$\begin{aligned}
(\Sigma\ \mathsf{sig})^{-\mathcal{UL}} &= \Sigma^{-\mathcal{UL}}\ \mathsf{sig}\\
(\vdash_\Sigma \Gamma)^{-\mathcal{UL}} &= \vdash_{\Sigma^{-\mathcal{UL}}} \Gamma^{-\mathcal{UL}},\\
(\Gamma \vdash_\Sigma K)^{-\mathcal{UL}} &= \Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} K^{-\mathcal{UL}},\\
(\Gamma \vdash_\Sigma \sigma : K)^{-\mathcal{UL}} &= \Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \sigma^{-\mathcal{UL}} : K^{-\mathcal{UL}},\\
(\Gamma \vdash_\Sigma M : \sigma)^{-\mathcal{UL}} &= \Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} M^{-\mathcal{UL}} : \sigma^{-\mathcal{UL}}.
\end{aligned}$$

With $^{-\mathcal{UL}}$ defined in this way, we have the following claim:

Proposition 1. *1. If $K{=}_{\beta\mathcal{L}}K'$ in $\mathsf{LF}_{\mathcal{P}}$, then $K^{-\mathcal{UL}}{=}_{\beta}K'^{-\mathcal{UL}}$ in LF.*
2. If $\sigma{=}_{\beta\mathcal{L}}\sigma'$ in $\mathsf{LF}_{\mathcal{P}}$, then $\sigma^{-\mathcal{UL}}{=}_{\beta}\sigma'^{-\mathcal{UL}}$ in LF.
3. If $M{=}_{\beta\mathcal{L}}M'$ in $\mathsf{LF}_{\mathcal{P}}$, then $M^{-\mathcal{UL}}{=}_{\beta}M'^{-\mathcal{UL}}$ in LF.

Furthermore, the following proposition holds:

Proposition 2. *The function $^{-\mathcal{UL}}$ maps derivable judgements of $\mathsf{LF}_{\mathcal{P}}$ into derivable judgements of LF.*

Proof. By induction on the structure of the derivation of the $\mathsf{LF}_{\mathcal{P}}$ judgement. □

Next, we will denote the maximum number of β-reductions which can be executed in a given (LF- or $\mathsf{LF}_{\mathcal{P}}$-) term T as $\max_\beta(T)$. Notice that $\mathcal{L}$-reductions cannot create entirely new β-redexes, but can only "unlock" potential β-redexes of the form $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\lambda x{:}\tau.M]]\,T$, arriving at $\lambda x{:}\tau.M\,T$. This redex will be present in $(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\lambda x{:}\tau.M]]\,T)^{-\mathcal{UL}}$. Therefore, we have that, for any $\mathsf{LF}_{\mathcal{P}}$-term T, it holds that $\max_\beta(T) \leq \max_\beta(T^{-\mathcal{UL}})$. As LF is strongly normalizing, we have that $\max_\beta(T^{-\mathcal{UL}})$ is finite, therefore forcing $\max_\beta(T)$ into being finite, leading to the following proposition:

Proposition 3. *Only finitely many β-reductions can occur within any $\mathsf{LF}_{\mathcal{P}}$-term.*

Next, we notice that any $\mathsf{LF}_{\mathcal{P}}$-term has only finitely many $\mathcal{L}$-redexes before any reductions take place, and that this number can be increased only through β-reductions, and only by a finite amount per β-reduction. However, if we were to have an $\mathsf{LF}_{\mathcal{P}}$-term T which has an infinite reduction sequence, then within this sequence, there would need to be infinitely many $\mathcal{L}$-reductions, since, due to Proposition 3, the number of β-reductions in this sequence has to be finite. On the other hand, with the number of β-reductions in the sequence being finite, it would not be possible to reach infinitely many $\mathcal{L}$-reductions, and such a term T cannot exist in $\mathsf{LF}_{\mathcal{P}}$. Therefore, we have the Strong Normalization theorem:

Theorem 9 (Strong normalization of $\mathsf{LF}_{\mathcal{P}}$). *1. If $\Gamma \vdash_\Sigma K$, then K is $\beta\mathcal{L}$-strongly normalizing.*
2. if $\Gamma \vdash_\Sigma \sigma : K$, then σ is $\beta\mathcal{L}$-strongly normalizing.
3. if $\Gamma \vdash_\Sigma M : \sigma$, then M is $\beta\mathcal{L}$-strongly normalizing.

Confluence.

Since $\beta\mathcal{L}$-reduction is strongly normalizing, in order to prove the confluence of the system, by *Newman's Lemma* ([3], Chapter 3), it is sufficient to show that the reduction on "raw terms" is *locally confluent*. First, we need a substitution lemma, whose proof is routine:

Lemma 1 (Substitution lemma for local confluence). *1. If $N \to_{\beta\mathcal{L}} N'$, then $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M[N'/x]$.*
2. If $M \to_{\beta\mathcal{L}} M'$, then $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$.

Hence we have Local Confluence:

Lemma 2 (Local confluence of $\mathsf{LF}_{\mathcal{P}}$). *$\beta\mathcal{L}$-reduction is locally confluent,* i.e.*:*
if $T \to_{\beta\mathcal{L}} T'$ and $T \to_{\beta\mathcal{L}} T''$, then there exists a T''', such that $T' \twoheadrightarrow_{\beta\mathcal{L}} T'''$ and $T'' \twoheadrightarrow_{\beta\mathcal{L}} T'''$.

Proof. By simultaneous induction on the two derivations $T \to_{\beta\mathcal{L}} T'$ and $T \to_{\beta\mathcal{L}} T''$. All the cases for T kind or family, as well as most of the cases for T object are proven trivially, using the induction hypotheses. Here we will show only the cases involving base reduction rules:

1. Let us have, by the base reduction rule $(\beta{\cdot}Main)$, $(\lambda x{:}\sigma.M)\,N \to_{\beta\mathcal{L}} M[N/x]$. Let us also have that $(\lambda x{:}\sigma.M)\,N \to_{\beta\mathcal{L}} (\lambda x{:}\sigma'.M)\,N$, from $\sigma \to_{\beta\mathcal{L}} \sigma'$, by the reduction rules $(O{\cdot}\lambda_1{\cdot}\beta\mathcal{L})$ and $(O{\cdot}App_1{\cdot}\beta\mathcal{L})$. In this case, we will show that the required conditions are met for $M''' \equiv M[N/x]$. Indeed, by the definition of $\twoheadrightarrow_{\beta\mathcal{L}}$, we have that $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M[N/x]$, and also, by the reduction rule $(\beta{\cdot}Main)$, we have that $(\lambda x{:}\sigma'.M)\,N \to_{\beta\mathcal{L}} M[N/x]$, effectively having $(\lambda x{:}\sigma'.M)\,N \twoheadrightarrow_{\beta\mathcal{L}} M[N/x]$.
2. Let us have, by the base reduction rule $(\beta{\cdot}Main)$, $(\lambda x{:}\sigma.M)\,N \to_{\beta\mathcal{L}} M[N/x]$. Let us also have that $(\lambda x{:}\sigma.M)\,N \to_{\beta\mathcal{L}} (\lambda x{:}\sigma.M')\,N$, from $M \to_{\beta\mathcal{L}} M'$, by the reduction rules $(O{\cdot}\lambda_2{\cdot}\beta\mathcal{L})$ and $(O{\cdot}App_1{\cdot}\beta\mathcal{L})$. In this case, we will show that the required conditions are met for $M''' \equiv M'[N/x]$. By $(\beta{\cdot}Main)$, we have $(\lambda x{:}\sigma.M')\,N \to_{\beta\mathcal{L}} M'[N/x]$, from which we obtain $(\lambda x{:}\sigma.M')\,N \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$, while we obtain that $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$ from part 2 of Lemma 1.
3. Let us have, by the base reduction rule $(\beta{\cdot}Main)$, $(\lambda x{:}\sigma.M)\,N \to_{\beta\mathcal{L}} M[N/x]$. Let us also have that $(\lambda x{:}\sigma.M)\,N \to_{\beta\mathcal{L}} (\lambda x{:}\sigma.M)\,N'$, from $N \to_{\beta\mathcal{L}} N'$, by the reduction rule $(O{\cdot}App_2{\cdot}\beta\mathcal{L})$. In this case, we will show that the required conditions are met for $M''' \equiv M[N'/x]$. By the reduction rule $(\beta{\cdot}Main)$, we have that $(\lambda x{:}\sigma.M)\,N' \to_{\beta\mathcal{L}} M[N'/x]$, from which we obtain $(\lambda x{:}\sigma.M)\,N' \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$, while we obtain that $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M[N'/x]$ from part 1 of Lemma 1.
4. Let us have, by the base reduction rule $(\mathcal{L}{\cdot}Main)$,$\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} M$, and, also, that $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]]$, from $N \to_{\beta\mathcal{L}} N'$, by the reduction rule $(O{\cdot}Unlock_1{\cdot}\beta\mathcal{L})$. In this case, we will show that the required conditions are met for $M''' \equiv M$. By the definition of $\twoheadrightarrow_{\beta\mathcal{L}}$, we have that $M \twoheadrightarrow_{\beta\mathcal{L}} M$, which leaves us with needing to show that $\mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \twoheadrightarrow_{\beta\mathcal{L}} M$. This we obtain by the following sequence of reductions: from $N \to_{\beta\mathcal{L}} N'$, which we have as an induction hypothesis, using the reduction rule $(O{\cdot}Lock_1{\cdot}\beta\mathcal{L})$, we obtain that $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N',\sigma}[M]$, and from this, using the reduction rule $(O{\cdot}Unlock_3{\cdot}\beta\mathcal{L})$, we obtain that $\mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N',\sigma}[M]]$, from which we finally obtain that $\mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N',\sigma}[M]] \to_{\beta\mathcal{L}} M$, by the reduction rule $(\mathcal{L}{\cdot}Main)$, effectively showing that $\mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \twoheadrightarrow_{\beta\mathcal{L}} M$. The remaining subcases are handled very similarly.

□

Having proven local confluence, finally, from Theorem 9, Lemma 2 and Newman's Lemma, we obtain the confluence theorem for $\mathsf{LF}_{\mathcal{P}}$:

Theorem 10 (Confluence of $\mathsf{LF}_{\mathcal{P}}$)**.** *$\beta\mathcal{L}$-reduction is confluent,* i.e.*: if $T \twoheadrightarrow_{\beta\mathcal{L}} T'$ and $T \twoheadrightarrow_{\beta\mathcal{L}} T''$, then there exists a T''', such that $T' \twoheadrightarrow_{\beta\mathcal{L}} T'''$ and $T'' \twoheadrightarrow_{\beta\mathcal{L}} T'''$.*

Subject reduction.

Lemma 3 (Auxiliary properties)**.**

Equivalence of products. *If $\Pi x{:}\sigma.T =_{\beta\mathcal{L}} T''$, then $T'' \equiv \Pi x{:}\sigma.'T'$, for some σ', T', such that $\sigma' =_{\beta\mathcal{L}} \sigma$, and $T' =_{\beta\mathcal{L}} T$.*

Equivalence of locks. *If $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] =_{\beta\mathcal{L}} \theta$, then $\theta \equiv \mathcal{L}^{\mathcal{P}}_{N',\sigma'}[\rho']$, for some N', σ', and ρ', such that $N' =_{\beta\mathcal{L}} N$, $\sigma' =_{\beta\mathcal{L}} \sigma$, and $\rho' =_{\beta\mathcal{L}} \rho$.*

Removal of locks. *If $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, then $\Gamma \vdash_\Sigma M : \rho$.*

The following property follows directly from the typing and conversion rules, using item 1 of Lemma 3:

Proposition 4 (Abstraction typing)**.** *If $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau$, then $\Gamma, x{:}\sigma \vdash_\Sigma M : \tau$.*

By induction on the structure of the derivation, we obtain:

Proposition 5 (Subderivation, part 1)**.**

1. *Any derivation of $\vdash_\Sigma \emptyset$ has a subderivation of Σ sig.*
2. *Any derivation of $\Sigma, a{:}K$ sig has subderivations of Σ sig and $\vdash_\Sigma K$.*
3. *Any derivation of $\Sigma, f{:}\sigma$ sig has subderivations of Σ sig and $\vdash_\Sigma \sigma{:}\mathsf{Type}$.*
4. *Any derivation of $\vdash_\Sigma \Gamma, x{:}\sigma$ has subderivations of Σ sig, $\vdash_\Sigma \Gamma$, and $\Gamma \vdash_\Sigma \sigma{:}\mathsf{Type}$.*
5. *Any derivation of $\Gamma \vdash_\Sigma \alpha$ has subderivations of Σ sig and $\vdash_\Sigma \Gamma$.*
6. *Given a derivation $\mathcal{D}$ 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.*

Proposition 6 (Weakening and permutation)**.** *If predicates are closed under signature/context weakening and permutation, then:*

1. *If Σ and Ω are valid signatures, and every declaration occurring in Σ also occurs in Ω, then $\Gamma \vdash_\Sigma \alpha$ implies $\Gamma \vdash_\Omega \alpha$.*
2. *If Γ and Δ are valid contexts w.r.t. the signature Σ, and every declaration occurring in Γ also occurs in Δ, then $\Gamma \vdash_\Sigma \alpha$ implies $\Delta \vdash_\Sigma \alpha$.*

Proposition 7 (Subderivation, part 2)**.** *If predicates are closed under signature/context weakening and permutation, then:*

1. *If $\Gamma \vdash_\Sigma \sigma : K$, then $\Gamma \vdash_\Sigma K$.*
2. *If $\Gamma \vdash_\Sigma M : \sigma$, then $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$.*

Proposition 8 (Transitivity)**.** *If predicates are closed under signature/context weakening and permutation and under substitution, then: if $\Gamma, x{:}\sigma, \Gamma' \vdash_\Sigma \alpha$, and $\Gamma \vdash_\Sigma N : \sigma$, then $\Gamma, \Gamma'[N/x] \vdash_\Sigma \alpha[N/x]$.*

Proposition 9 (Unicity of types and kinds)**.** *If predicates are closed under signature/context weakening and permutation and under substitution, then: if $\Gamma \vdash_\Sigma T : T_1$ and $\Gamma \vdash_\Sigma T : T_2$, then $\Gamma \vdash_\Sigma T_1 =_{\beta\mathcal{L}} T_2$.*

Finally, we have Subject Reduction:

Theorem 11 (Subject reduction of $\mathsf{LF}_{\mathcal{P}}$)**.** *If predicates are well-behaved, then:*

1. *If $\Gamma \vdash_\Sigma K$, and $K \rightarrow_{\beta\mathcal{L}} K'$, then $\Gamma \vdash_\Sigma K'$.*
2. *If $\Gamma \vdash_\Sigma \sigma : K$, and $\sigma \rightarrow_{\beta\mathcal{L}} \sigma'$, then $\Gamma \vdash_\Sigma \sigma' : K$.*
3. *If $\Gamma \vdash_\Sigma M : \sigma$, and $M \rightarrow_{\beta\mathcal{L}} M'$, then $\Gamma \vdash_\Sigma M' : \sigma$.*

Proof. Here we prove Subject Reduction of a slightly extended type system. We consider the type system in which the rules $(F{\cdot}Lock)$, $(O{\cdot}Lock)$, and $(O{\cdot}Unlock)$ all have an additional premise $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$, while the rule $(O{\cdot}Unlock)$ also has another additional premise $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$, as shown in Figure 14.

$$\frac{\Gamma \vdash_\Sigma \rho : \mathsf{Type} \quad \Gamma \vdash_\Sigma N : \sigma \quad \Gamma \vdash_\Sigma \sigma : \mathsf{Type}}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}} \ (\mathsf{F{\cdot}Lock})$$

$$\frac{\Gamma \vdash_\Sigma M : \rho \quad \Gamma \vdash_\Sigma N : \sigma \quad \Gamma \vdash_\Sigma \sigma : \mathsf{Type}}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]} \ (\mathsf{O{\cdot}Lock})$$

$$\frac{\begin{array}{c}\Gamma \vdash_\Sigma N : \sigma \quad \Gamma \vdash_\Sigma \sigma : \mathsf{Type} \quad \mathcal{P}(\Gamma \vdash_\Sigma N : \sigma) \\ \Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \quad \Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}\end{array}}{\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho} \ (\mathsf{O{\cdot}Unlock})$$

Figure 14. An extension of $\mathsf{LF}_{\mathcal{P}}$ typing rules required for Subject Reduction

The proof proceeds by simultaneous induction on the derivation of $\Gamma \vdash_\Sigma M$ and $M \rightarrow_{\beta\mathcal{L}} M'$. Here we will show only the cases in which the base reduction rules are used, and one of the cases for which the well-behavedness of predicates is a requirement, while the other cases are handled either similarly or trivially, mostly by using the induction hypotheses.

1. We have that $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M\,N : \tau[N/x]$, by the type system rule $(O{\cdot}App)$, from $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau$, and $\Gamma \vdash_\Sigma N : \sigma$, and that $(\lambda x{:}\sigma.M)\,N \rightarrow_{\beta\mathcal{L}} M[N/x]$ by the reduction rule $(\beta{\cdot}Main)$. From Proposition 4, we get that $\Gamma, x{:}\sigma \vdash_\Sigma M : \tau$, and from this and $\Gamma \vdash_\Sigma N : \sigma$, we obtain the required $\Gamma \vdash_\Sigma M[N/x] : \tau[N/x]$, by an application of Proposition 8.
2. We have that $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] : \rho$, by the type system rule $(O{\cdot}Unlock)$, from $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, $\Gamma \vdash_\Sigma N : \sigma$, $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$, and $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$, and that $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \rightarrow_{\beta\mathcal{L}} M$ by the reduction rule $(\mathcal{L}{\cdot}Main)$,. Here, we obtain the required $\Gamma \vdash_\Sigma M : \rho$ directly, using the last two items of Lemma 3.
3. We have that $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho$, by the type system rule $(O{\cdot}Unlock)$, from $\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$, $\Gamma \vdash_\Sigma N : \sigma$, $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$, and $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$, and that $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \rightarrow_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N,\sigma'}[M]$, by the reduction rule $(O{\cdot}Unlock_2)$, from $\sigma \rightarrow_{\beta\mathcal{L}} \sigma'$. First, from the induction hypothesis we have that $\Gamma \vdash_\Sigma \sigma' : \mathsf{Type}$, and we also have, from $\sigma \rightarrow_\beta{:}\ \sigma'$, that $\sigma =_{\beta\mathcal{L}} \sigma'$. From this, using $\Gamma \vdash_\Sigma N : \sigma$, and the type system rule $(O{\cdot}Conv)$, we obtain that $\Gamma \vdash_\Sigma N : \sigma'$. Next, since $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$ could only have been obtained by the type system rule $(F{\cdot}Lock)$, from $\Gamma \vdash_\Sigma \rho : \mathsf{Type}$ and $\Gamma \vdash_\Sigma N : \sigma$, and since we have $\Gamma \vdash_\Sigma N : \sigma'$, we obtain that $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho] : \mathsf{Type}$. From this, given $\sigma =_{\beta\mathcal{L}} \sigma'$, using the reduction rule $(F{\cdot}Lock_2{\cdot}\beta\mathcal{L})$, we obtain that $\mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho] \rightarrow_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho]$, and since we already have that $\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, we can use the type system rule $(O{\cdot}Conv)$ to obtain $\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho]$. Finally, by the well-behavedness requirements for the predicates, we have that $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma')$ holds, and we can now use the type system rule $(O{\cdot}Unlock)$ to obtain the required $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma'}[M] : \rho$. Here, we can notice that there are steps in this proof (in which we obtain $\Gamma \vdash_\Sigma \sigma' : \mathsf{Type}$, and $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$), which could not have been made had the original system not been extended for this theorem.

Now, we can prove straightforwardly that $\Gamma \vdash_\Sigma \alpha$ in the extended system *if and only if* $\Gamma \vdash_\Sigma \alpha$ in the original $\mathsf{LF}_{\mathcal{P}}$ system (i. e. that the judgements that these two systems derive are, in fact, the same), by induction on the length of the derivation, With this, given that we have proven Subject Reduction of the extended system, we obtain that Subject Reduction also holds in the original $\mathsf{LF}_{\mathcal{P}}$ system. ☐

A.2 Proofs of adequacy theorems

A.2.1 Proof of Theorem 2.

Proof. The injectivity of $\epsilon_{\mathcal{X}}$ follows by a straightforward inspection of its definition, while the surjectivity follows by defining the "decoding" function $\delta_{\mathcal{X}}$:

$$\begin{aligned}
\delta_{\mathcal{X}}(\texttt{(free i)}) &= x_i \\
\delta_{\mathcal{X}}(\texttt{xi}) &= x_i \\
\delta_{\mathcal{X}}(\texttt{(app M N)}) &= \delta_{\mathcal{X}}(\texttt{M})\,\delta_{\mathcal{X}}(\texttt{N}) \\
\delta_{\mathcal{X}}(\texttt{lambda M}) &= \lambda x.\delta_{\mathcal{X}\cup\{x\}}(\texttt{M x})
\end{aligned}$$

Given the characterisation of the η-long normal forms and the types of the constructors introduced in Σ_λ, it is easy to see that $\delta_{\mathcal{X}}$ is total and well defined. Notice that it is not possible to derive a η-long normal form of type `term` containing a $\mathcal{U}$-term, since no constructors in Σ_λ use $\mathcal{L}$-types. Finally, by induction on the structure of M, it is possible to check that $\delta_{\mathcal{X}}(\epsilon_{\mathcal{X}}(M)) = M$ and that $\epsilon_{\mathcal{X}}$ is compositional. □

A.2.2 Proof of Theorem 3.

Proof. We define an encoding function $\epsilon_{\emptyset}^{=}$ by induction on derivations of the form $\vdash_{CBV} M = N$ (on terms with no bindable variables) as follows:

- if ∇ is the derivation
$$\frac{}{\vdash_{CBV} M = M}$$
then $\epsilon_{\emptyset}^{=}(\nabla) = (\texttt{refl}\ \epsilon_\emptyset(M))\texttt{:}(\texttt{eq}\ \epsilon_\emptyset(M)\ \epsilon_\emptyset(M))$;
- if ∇ is the derivation with (symm) as last applied rule, then, by inductive hypothesis, there is a term `h` such that $\vdash_{\Sigma_{CBV}} \texttt{h} : (\texttt{eq}\ \epsilon_\emptyset(N)\ \epsilon_\emptyset(M))$. Hence, $\epsilon_{\emptyset}^{=}(\nabla) = (\texttt{symm}\ \epsilon_\emptyset(M)\ \epsilon_\emptyset(N)\ \texttt{h})\texttt{:}(\texttt{eq}\ \epsilon_\emptyset(M)\ \epsilon_\emptyset(N))$;
- if ∇ is the derivation with (trans) as last applied rule, then, by inductive hypothesis, there are terms `h` and `h'` such that $\vdash_{\Sigma_{CBV}} \texttt{h} : (\texttt{eq}\ \epsilon_\emptyset(M)\ \epsilon_\emptyset(N))$ and $\vdash_{\Sigma_{CBV}} \texttt{h}' : (\texttt{eq}\ \epsilon_\emptyset(N)\ \epsilon_\emptyset(P))$. Hence, $\epsilon_{\emptyset}^{=}(\nabla) = (\texttt{trans}\ \epsilon_\emptyset(M)\ \epsilon_\emptyset(N)\ \epsilon_\emptyset(P)\ \texttt{h}\,\texttt{h}')\texttt{:}(\texttt{eq}\ \epsilon_\emptyset(M)\ \epsilon_\emptyset(P))$;
- if ∇ is the derivation with $(\mathsf{eq\_app})$ as last applied rule, then, by inductive hypothesis, there are terms `h` and `h'` such that $\vdash_{\Sigma_{CBV}} \texttt{h} : (\texttt{eq}\ \epsilon_\emptyset(M)\ \epsilon_\emptyset(N))$ and $\vdash_{\Sigma_{CBV}} \texttt{h}' : (\texttt{eq}\ \epsilon_\emptyset(M')\ \epsilon_\emptyset(N'))$. Thus, $\epsilon_{\emptyset}^{=}(\nabla) = (\texttt{eq\_app}\ \epsilon_\emptyset(M)\ \epsilon_\emptyset(N)\ \epsilon_\emptyset(M')\ \epsilon_\emptyset(N')\ \texttt{h}\,\texttt{h}')$: $(\texttt{eq}\ (\texttt{app}\ \epsilon_\emptyset(M)\ \epsilon_\emptyset(M'))\ (\texttt{app}\ \epsilon_\emptyset(N)\ \epsilon_\emptyset(N')))$;
- if ∇ is the derivation
$$\frac{v \text{ is a value}}{\vdash_{CBV} (\lambda x.M)v = M[v/x]}$$
then
$\epsilon_{\emptyset}^{=}(\nabla) = \mathcal{U}^{\mathit{Val}}_{\epsilon_\emptyset(v),\texttt{term}}[(\texttt{betav}\ (\lambda\texttt{x} : \texttt{term}.\epsilon_{\{x\}}(M))\ \epsilon_\emptyset(v))]$: $(\texttt{eq}\ (\texttt{app}\ (\texttt{lambda}\ \lambda\texttt{x} : \texttt{term}.\epsilon_{\{x\}}(M))\ \epsilon_\emptyset(v))\ ((\lambda\texttt{x} : \texttt{term}.\epsilon_{\{x\}}(M))(\epsilon_\emptyset(v))))$ (notice the presence of the unlock operator in front of the $\mathsf{LF}_{\mathcal{P}}$ encoding: this is possible thanks to the fact that we know by hypothesis that v is a value, whence the predicate Val holds on $\vdash_{CBV} \epsilon_\emptyset(v) : \texttt{term}$);
- if ∇ is the derivation with (ξ_v) as last applied rule, then, by inductive hypothesis, there is a term `h` such that $\vdash_{\Sigma_{CBV}} \texttt{h} : (\texttt{eq}\ \epsilon_\emptyset(M)\ \epsilon_\emptyset(N))^2$. So,
$\epsilon_{\emptyset}^{=}(\nabla) = (\mathcal{U}^{\xi}_{\texttt{T},\texttt{triple}}[(\texttt{csiv}\ \lambda\texttt{x:term}.\epsilon_{\{x\}}(M)\ \lambda\texttt{x:term}.\epsilon_{\{x\}}(N)\ \epsilon_\emptyset(x))]$ $\texttt{h})$: $(\texttt{eq}\ (\texttt{lambda}\ \lambda\texttt{x:term}.\ \epsilon_{\{x\}}(M))\ (\texttt{lambda}\ \lambda\texttt{x:term}.\epsilon_{\{x\}}(N)))$, where `T` is the triple $\langle \epsilon_\emptyset(x), (\lambda\texttt{x:term}.\epsilon_{\{x\}}(M)), (\lambda\texttt{x:term}.\epsilon_{\{x\}}(N))\rangle$.

The injectivity of $\epsilon_{\emptyset}^{=}$ follows by a straightforward inspection of its definition, while the surjectivity follows by defining the "decoding" function $\delta_\emptyset$ by induction on the derivations of the shape $\vdash_{\Sigma_{CBV}} \texttt{h:(eq M N)}$ in η-long normal form. Since all the cases are rather straightforward, we analyze only the definition concerning the main rule (β_v), since it involves an external predicate. So, if we derive from Σ_{CBV} a proof term `h` in η-long normal form such as $\mathcal{U}^{\mathit{Val}}_{\texttt{N},\texttt{term}}[\texttt{betav M N}]\texttt{:(eq (app (lambda M) N) (M N))}$, then the predicate $\mathit{Val}(\vdash_{\Sigma_{CBV}} \texttt{N} : \texttt{term})$ must hold, hence `N` is encoding the value $\delta_\emptyset(\texttt{N})$. Hence, the *decoding* of `h` is the following derivation:

$$\frac{\delta_\emptyset(\texttt{N}) \text{ is a value}}{\vdash_{CBV} \delta_\emptyset(\texttt{(lambda M)})\delta_\emptyset(\texttt{N}) = \delta_\emptyset(\texttt{M N})}$$

since $\delta_\emptyset(\texttt{(lambda M)}) \equiv \lambda x.\delta_{\{x\}}(\texttt{(M x)})$ and $\delta_\emptyset(\texttt{M N}) \equiv \delta_{\{x\}}(\texttt{M x})[\delta_\emptyset(\texttt{N})/x]$ (by induction on the structure of `M`), we are done. Therefore, it is easy to verify by induction on η-long normal forms that $\delta_{\emptyset}^{=}$ is well defined and total. Another easy induction proves that $\delta_{\emptyset}^{=}$ is the inverse function of $\epsilon_{\emptyset}^{=}$, so the latter is bijective. □

[2] Notice that the object variable x occurring in M and N is represented by a constant (`(free k)` for the natural k such that $x \equiv x_k$) here, since the encoding function takes the empty set as the set of *bindable* variables. Instead, in the next line, the encoding function will take $\{x\}$ as the set of bindable variables, yielding an encoding of x through a metavariable `x` of the metalanguage of $\mathsf{LF}_{\mathcal{P}}$.