LFP – A Logical Framework with External Predicates (Announcement and survey)

Furio Honsell, Marina Lenisa, Luigi Liquori, Petar Maksimovic, Ivan Scagnetto

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

Furio Honsell[1], Marina Lenisa[1], Luigi Liquori[2], Petar Maksimovic[2,3], Ivan Scagnetto[1]

[1] Università di Udine, Italy
[furio.honsell, marina.lenisa, ivan.scagnetto]@uniud.it
[2] Institut National de Recherche en Informatique et en Automatique, France
Luigi.Liquori@inria.fr
[3] Mathematical Institute of the Serbian Academy of Sciences and Arts, Serbia
petarmax@mi.sanu.ac.rs

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 $\mathsf{LF}_{\mathcal{P}}$, together with its *canonical* presentation, based on a suitable extension of the notion of $\beta\eta$-*long normal form*, allowing for smooth formulation of *adequacy* statements.

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 an appropriate *introduction rule*. Correspondingly, we introduce an unlock *destructor*, $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]$, and an *elimination rule*, 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. The *equality rule* for lock types amounts to a lock reduction ($\mathcal{L}$-reduction),

* Work supported by Serbian Ministry of Education and Science (projects ON 174026, III 044006) and Italian Ministry of Education, University and Research.

$\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\mathcal{L}} M$, 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*, *substitution*, and *$\beta\mathcal{L}$-reduction* in the arguments. $\mathsf{LF}_{\mathcal{P}}$ is decidable, if the external predicates are decidable. We also provide a *canonical* presentation of $\mathsf{LF}_{\mathcal{P}}$, in the style of [20,12], based on a suitable extension of the notion of $\beta\eta$-long normal form. This allows for simple proofs of adequacy of the encodings.

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, both in Hilbert and Natural Deduction style, *cf.* [2,8]. We also encode substructural logics, including non-commutative Linear Logic, *cf.* [19,8]. $\mathsf{LF}_{\mathcal{P}}$ 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 [7,5]. 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 [7,5,13,14], 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 [17], 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 [18], 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 [15], but more work needs to be done to compare precisely the expressive powers.

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 meta-theoretical properties of the system. In Section 3, we present a canonical version of $\mathsf{LF}_{\mathcal{P}}$, and we discuss the correspondence with the full framework. In Section 4, we show how to encode call-by-value λ-calculus, Modal Logics, and non-commutative Linear Logic. Conclusions and future work appear in Section 5. In Appendix A, we collect complete definitions and proofs of the properties of $\mathsf{LF}_{\mathcal{P}}$, while in Appendix B details about the examples appear, together with a library of auxiliary functions, and encodings of program logics à la Hoare.

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.

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

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

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

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

In a typing judgement $\Gamma \vdash_\Sigma T : T'$ ($\Gamma \vdash_\Sigma T$), 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 Appendix A 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 definitional equality*). All of the above conditions on predicates are collected in the definition of *well-behaved predicates*:

Signature rules

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

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

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

Context rules

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

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

Kind rules

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

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

Family rules

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

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

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

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

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

Object rules

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

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

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

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

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

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

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

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

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 and 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 definitional equality. *If* $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$ *holds and* $N \to_{\beta\mathcal{L}} N'$ *(*$\sigma \to_{\beta\mathcal{L}} \sigma'$*) holds, then* $\mathcal{P}(\Gamma \vdash_\Sigma N' : \sigma)$ *(*$\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma')$*) 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 a 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* $\Gamma \vdash_\Sigma M' : \sigma$.

$$(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x] \quad (\beta{\cdot}Main)$$
$$\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} M \quad (\mathcal{L}{\cdot}Main)$$

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

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, for lack of space, 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 4.2 below). The former essentially correspond to the internal encoding of the predicates that are utilized in Section 4.2. 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 definitional equality, 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 4.2, 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 The Canonical $\mathsf{LF}_{\mathcal{P}}$ Framework

In this section, we present a *canonical* version of $\mathsf{LF}_{\mathcal{P}}(\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}})$, in the style of [20,12]. This amounts to an extension of the standard η-rule with the clause $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]] \rightarrow_{\eta} M$, corresponding to the lock type constructor. The syntax of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ defines the *normal forms* of $\mathsf{LF}_{\mathcal{P}}$, and the typing system captures all of the judgements in *η-long normal form* which are derivable in $\mathsf{LF}_{\mathcal{P}}$. $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ will be the basis for proving the adequacy of the encodings of Section 4. As will be seen, contrary to standard LF, *not all* of the judgements derivable in $\mathsf{LF}_{\mathcal{P}}$ admit a corresponding η-long normal form. In fact, this is not the case when the predicates appearing in the $\mathsf{LF}_{\mathcal{P}}$ judgement are not satisfied in the given context. Nevertheless, although $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ is not closed under full η-expansion, it is powerful enough for one to be able to obtain all relevant adequacy results.

$\Sigma \in \mathcal{S}$	$\Sigma ::= \emptyset \mid \Sigma, a{:}K \mid \Sigma, c{:}\sigma$	*Signatures*
$\Gamma \in \mathcal{C}$	$\Gamma ::= \emptyset \mid \Gamma, x{:}\sigma$	*Contexts*
$K \in \mathcal{K}$	$K ::= \mathsf{Type} \mid \Pi x{:}\sigma.K$	*Kinds*
$\alpha \in \mathcal{A}_f$	$\alpha ::= a \mid \alpha\, N$	*Atomic Families*
$\sigma, \tau, \rho \in \mathcal{F}$	$\sigma ::= \alpha \mid \Pi x{:}\sigma.\tau \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$	*Canonical Families*
$A \in \mathcal{A}_o$	$A ::= c \mid x \mid A\, M \mid \mathcal{U}^{\mathcal{P}}_{N,\sigma}[A]$	*Atomic Objects*
$M, N \in \mathcal{O}$	$M ::= A \mid \lambda x{:}\sigma.M \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]$	*Canonical Objects*

Fig. 4. $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ Syntax

Before we proceed to present the type system for $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, we introduce the notion of *hereditary substitution*, which computes the normal form resulting from the substitution of one normal form into another. The general form of the hereditary substitution judgement is $T[M/x]^m_\rho = T'$, where M is the term being substituted, x is the variable being substituted for, T is the term being substituted into, T' is the result of the substitution, ρ is the simple type of M, and m is the syntactic category being involved in the judgement (*i.e.* kinds, atomic/canonical families, atomic/canonical objects, contexts). For lack of space, here we present only some of the most important rules for the hereditary substitution judgement, in Figure 5. The mapping of dependent into simple types, the full set of hereditary substitution rules, together with proofs of the properties of hereditary substitution and of the type system, appear in Appendix A.

Lemma 1 (Decidability of hereditary substitution).
For any T in $\{\mathcal{K}, \mathcal{A}_f, \mathcal{F}, \mathcal{O}, \Gamma\}$, and any M, x, and ρ, either there exists a T' such that $T[M/x]^m_\rho = T'$ or there is no such T'.
For any M, x, ρ, and A, either there exists A' such that $A[M/x]^o_\rho = A'$, or there exist M' and ρ', such that $A[M/x]^o_\rho = M' : \rho'$, or there are no such A' or M'.

3.1 Type system for $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$

The pseudo-syntax of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ is presented in Figure 4, while the type system for $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, shown in Figure 6, proves judgements of the shape:

$\Sigma\ \mathsf{sig}$	Σ is a valid signature
$\vdash_\Sigma \Gamma$	Γ is a valid context in Σ
$\Gamma \vdash_\Sigma K$	K is a kind in Γ and Σ
$\Gamma \vdash_\Sigma \sigma\ \mathsf{Type}$	σ is a canonical family in Γ and Σ
$\Gamma \vdash_\Sigma \alpha \Rightarrow K$	K is the kind of the atomic family α in Γ and Σ
$\Gamma \vdash_\Sigma M \Leftarrow \sigma$	M is a canonical term of type σ in Γ and Σ
$\Gamma \vdash_\Sigma A \Rightarrow \sigma$	σ is the type of the atomic term A in Γ and Σ

The predicates $\mathcal{P}$ in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ are defined on judgements $\Gamma \vdash_\Sigma M \Leftarrow \sigma$; we refer to a predicate $\mathcal{P}$ as *well-behaved* if it is *closed under signature* and *context weakening* and *permutation*, and *hereditary substitution*.

In the type system, the rules ($A{\cdot}App$) and ($F{\cdot}App$) have hereditary substitution premises which compute the resulting type or kind for applications. In the rule ($O{\cdot}Atom$), the syntactic restriction of the classifier to α atomic, ensures that canonical forms are *η-long normal forms* for a suitable notion of η-long normal form, which extends the standard one for lock-types. For one, the judgement $x{:}\Pi z{:}a.a \vdash_\Sigma x \Leftarrow \Pi z{:}a.a$ is not derivable, as $\Pi z{:}a.a$ is not atomic, hence $\vdash_\Sigma \lambda x{:}(\Pi z{:}a.a).x \Leftarrow \Pi x{:}(\Pi z{:}a.a).\Pi z{:}a.a$ is not derivable. But $\vdash_\Sigma \lambda x{:}(\Pi z{:}a.a).\lambda y{:}a.xy \Leftarrow \Pi x{:}(\Pi z{:}a.a).\Pi z{:}a.a$, where a is a family constant of kind *Type*, is derivable. Analogously, for lock-types, the judgement $x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \vdash_\Sigma x \Leftarrow \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$ is not derivable, since $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$ is not atomic. As a consequence, $\vdash_\Sigma \lambda x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho].x \Leftarrow \Pi x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho].\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$ is not derivable. However, $x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{U}^{\mathcal{P}}_{N,\sigma}[x]] \Leftarrow \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$ is derivable, if ρ is atomic and the predicate $\mathcal{P}$ holds on $\Gamma \vdash_\Sigma N \Leftarrow \sigma$. Hence $\vdash_\Sigma \lambda x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho].\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{U}^{\mathcal{P}}_{N,\sigma}[x]] \Leftarrow \Pi x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho].\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$ is derivable. We will formalize the notion of η-expansion of a judgement in Definition 3, together with correspondence theorems between the original $\mathsf{LF}_{\mathcal{P}}$ system and $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$.

Theorem 2 (Decidability of typing). *If predicates in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ are decidable, then all of the judgements of the system are decidable.*

Theorem 3 (Soundness). *For any predicate $\mathcal{P}$ of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, we define a corresponding predicate in $\mathsf{LF}_{\mathcal{P}}$ with: $\mathcal{P}(\Gamma \vdash_\Sigma M : \sigma)$ holds if and only if $\Gamma \vdash_\Sigma M : \sigma$ is derivable in $\mathsf{LF}_{\mathcal{P}}$ and $\mathcal{P}(\Gamma \vdash_\Sigma M \Leftarrow \sigma)$ holds in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$. Then, we have:*

1. *If $\Sigma\ \mathsf{sig}$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\Sigma\ \mathsf{sig}$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
2. *If $\vdash_\Sigma \Gamma$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\vdash_\Sigma \Gamma$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
3. *If $\Gamma \vdash_\Sigma K$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\Gamma \vdash_\Sigma K$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
4. *If $\Gamma \vdash_\Sigma \alpha \Rightarrow K$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\Gamma \vdash_\Sigma \alpha : K$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
5. *If $\Gamma \vdash_\Sigma \sigma\ \mathsf{Type}$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
6. *If $\Gamma \vdash_\Sigma A \Rightarrow \sigma$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\Gamma \vdash_\Sigma A : \sigma$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
7. *If $\Gamma \vdash_\Sigma M \Leftarrow \sigma$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\Gamma \vdash_\Sigma M : \sigma$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*

Vice versa, all $\mathsf{LF}_{\mathcal{P}}$ judgements which are in *η-long normal form* (η-lnf) are derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$. The definition of a judgement in η-lnf is based on the extension of the standard η-rule to the lock constructor as follows:

$$\lambda x{:}\sigma.Mx \to_\eta M \qquad \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]] \to_\eta M\ .$$

Definition 2. *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.*

Substitution in Atomic Families

$$\frac{}{a[M/x]^f_\rho = a}\ (\mathcal{S}{\cdot}F{\cdot}Const) \qquad \frac{\alpha[M/x]^f_\rho = \alpha' \quad N[M/x]^O_\rho = M'}{\alpha N[M/x]^f_\rho = \alpha' N'}\ (\mathcal{S}{\cdot}F{\cdot}App)$$

Substitution in Atomic Objects

$$\frac{AN[M/x]^o_\rho = \lambda x{:}\rho_1.M_1 : \rho_1 \to \rho' \quad N[M/x]^O_\rho = N' \quad M_1[N'/x]^O_{\rho_1} = M'}{AN[M/x]^o_\rho = M' : \rho'}\ (\mathcal{S}{\cdot}O{\cdot}App{\cdot}H)$$

$$\frac{\sigma[M/x]^F_\rho = \sigma' \quad N[M/x]^O_\rho = N' \quad A[M/x]^o_\rho = \mathcal{L}^{\mathcal{P}}_{N',\sigma'}[M'] : \mathcal{L}^{\mathcal{P}}_{N',\sigma'}[\rho']}{\mathcal{U}^{\mathcal{P}}_{N,\sigma}[A][M/x]^o_\rho = M' : \rho'}\ (\mathcal{S}{\cdot}O{\cdot}Unlock{\cdot}H)$$

Substitution in Canonical Objects

$$\frac{A[M/x]^o_\rho = A'}{A[M/x]^O_\rho = A'}\ (\mathcal{S}{\cdot}O{\cdot}R) \qquad \frac{A[M/x]^o_\rho = M' : \rho}{A[M/x]^O_\rho = M'}\ (\mathcal{S}{\cdot}O{\cdot}R{\cdot}H)$$

Fig. 5. Hereditary substitution rules.

Definition 3 (Judgements in η-long normal form).
- 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.*

Theorem 4 (Correspondence). *Assume that all predicates in $\mathsf{LF}_{\mathcal{P}}$ are well-behaved. For any predicate $\mathcal{P}$ in $\mathsf{LF}_{\mathcal{P}}$, we define a corresponding predicate in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ with: $\mathcal{P}(\Gamma \vdash_\Sigma M \Leftarrow \sigma)$ holds if $\Gamma \vdash_\Sigma M \Leftarrow \sigma$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ and $\mathcal{P}(\Gamma \vdash_\Sigma M : \sigma)$ holds in $\mathsf{LF}_{\mathcal{P}}$. Then, we have:*

1. *If Σ sig is in η-lnf, and is $\mathsf{LF}_{\mathcal{P}}$-der, then Σ sig is $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$-der.*
2. *If $\vdash_\Sigma \Gamma$ is in η-lnf, and is $\mathsf{LF}_{\mathcal{P}}$-der, then $\vdash_\Sigma \Gamma$ is $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$-der.*
3. *If $\Gamma \vdash_\Sigma K$ is in η-lnf, and is $\mathsf{LF}_{\mathcal{P}}$-der, then $\Gamma \vdash_\Sigma K$ is $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$-der.*
4. *If $\Gamma \vdash_\Sigma \alpha : K$ is in η-lnf, and is $\mathsf{LF}_{\mathcal{P}}$-der, then $\Gamma \vdash_\Sigma \alpha \Rightarrow K$ is $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$-der.*
5. *If $\Gamma \vdash_\Sigma \sigma{:}\mathsf{Type}$ is in η-lnf, and is $\mathsf{LF}_{\mathcal{P}}$-der, then $\Gamma \vdash_\Sigma \sigma$ Type is $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$-der.*
6. *If $\Gamma \vdash_\Sigma A : \alpha$ is in η-lnf, and is $\mathsf{LF}_{\mathcal{P}}$-der, then $\Gamma \vdash_\Sigma A \Rightarrow \alpha$ is $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$-der.*
7. *If $\Gamma \vdash_\Sigma M : \sigma$ is in η-lnf, and is $\mathsf{LF}_{\mathcal{P}}$-der, then $\Gamma \vdash_\Sigma M \Leftarrow \sigma$ is $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$-der.*

Notice that, by the Correspondence Theorem above, any well-behaved predicate $\mathcal{P}$ in $\mathsf{LF}_{\mathcal{P}}$ induces a well-behaved predicate in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$. Finally, notice that *not* all $\mathsf{LF}_{\mathcal{P}}$ judgements have a corresponding η-long normal form. Namely, the judgement $x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \vdash_\Sigma x : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$ does not admit an η-expanded normal form when the predicate $\mathcal{P}$ does *not* hold on N, since the rule $(O{\cdot}Unlock)$ can be applied only when the predicate holds.

Signature rules

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

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

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

Context rules

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

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

Kind rules

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

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

Atomic Family rules

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

$$\frac{\Gamma \vdash_\Sigma \alpha \Rightarrow \Pi x{:}\sigma.K_1 \quad \Gamma \vdash_\Sigma M \Leftarrow \sigma \quad K_1[M/x]^K_\sigma = K}{\Gamma \vdash_\Sigma \alpha\, M \Rightarrow K}\ (A{\cdot}App)$$

Canonical Family rules

$$\frac{\Gamma \vdash_\Sigma \alpha \Rightarrow \mathsf{Type}}{\Gamma \vdash_\Sigma \alpha\ \mathsf{Type}}\ (F{\cdot}Atom)$$

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

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

Atomic Object rules

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

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

$$\frac{\Gamma \vdash_\Sigma A \Rightarrow \Pi x{:}\sigma.\tau_1 \quad \Gamma \vdash_\Sigma M \Leftarrow \sigma \quad \tau_1[M/x]^F_\sigma = \tau}{\Gamma \vdash_\Sigma A\, M \Rightarrow \tau}\ (O{\cdot}App)$$

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

Canonical Object rules

$$\frac{\Gamma \vdash_\Sigma A \Rightarrow \alpha}{\Gamma \vdash_\Sigma A \Leftarrow \alpha}\ (O{\cdot}Atom)$$

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

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

Fig. 6. The $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ Type System

4 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 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. We do not pursue this, for lack of space.

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 [19]. In Appendix 7.4, we present a basic library of auxiliary functions, which can be used to introduce external predicates of the above archetypes. Another example, on program logics *à la* Hoare appears in Appendix 7.6. The adequacy proofs exploit the canonical system $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, in a standard way. For lack of space and since it is routine, we state the adequacy theorems only for λ-calculus.

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 $\sigma \to \ldots \to \sigma$.

4.1 The untyped λ-calculus

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.

Definition 4 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ_λ for untyped λ-calculus).

```
nat, term : Type          0 : nat          S : nat^2
    free : nat -> term   app : term^3   lambda : term^2 -> term
```

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: $\epsilon_{\mathcal{X}}(x_i) = \begin{cases} \texttt{xi}, \text{ if } x_i \in \mathcal{X} \\ \texttt{free(i)}, \text{ if } x_i \notin \mathcal{X} \end{cases}$, $\epsilon_{\mathcal{X}}(MN) = (\texttt{app}\ \epsilon_{\mathcal{X}}(M)\,\epsilon_{\mathcal{X}}(N))$, and $\epsilon_{\mathcal{X}}(\lambda x.M) = (\texttt{lambda}\ \lambda\texttt{x:term}.\epsilon_{\mathcal{X}\cup\{\texttt{x}\}}(M))$, where in the latter clause, $x \notin \mathcal{X}$.

Theorem 5 (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 $\Gamma \vdash_{\Sigma_\lambda} M : \texttt{term}$ in η-lnf, where $\Gamma = \{\texttt{x} : \texttt{term} \mid x \in \mathcal{X}\}$. Moreover, the encoding is compositional, i.e. for a term $M[x_1, \ldots, x_k]$, 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:*

$$\epsilon_{\mathcal{X}}(M[N_1, \ldots, N_k]) = [\epsilon_{\mathcal{Y}}(N_1), \ldots, \epsilon_{\mathcal{Y}}(N_k)/x_1, \ldots, x_k]\epsilon_{\mathcal{X}}(M).$$

Untyped λ-calculus and call-by-value reduction strategy. The call-by-value (CBV) evaluation strategy can be specified by the rule $(\lambda x.M)\ v \to M[v/x]$ where v is a value, *i.e.* , a variable (constant in our encoding) or a function.

Definition 5 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ_{CBV} for λ-calculus definitional equality). *We extend the signature of Definition 4 as follows (due to lack of space, the formal system of definitional equality and its full encoding appear in Appendix 7.2):*
`betav : ΠM:term`^{2}`.ΠN:term.`$\mathcal{L}^{\mathrm{Val}}_{\mathtt{N}}$`[(eq (app (lambda M) N) (M N))]`
`csiv : ΠM,N:term`^{2}`.Πx:term.`
$\mathcal{L}^{\xi}_{\langle \mathtt{x},\mathtt{M},\mathtt{N}\rangle,\mathtt{triple}}$`[(eq (M x) (N x)) -> (eq (lambda M) (lambda N))]`
where the predicates ***Val*** *, ξ are defined as follows and* `triple` *is the obvious type of triples of terms with types* `term`*,* `term`^{2} *and* `term`^{2}*:*
- ***Val*** $(\Gamma \vdash_{\Sigma}$ `N:term`$)$ *holds iff either* `N` *is an abstraction or* `N` *is a constant (i.e. a term of the shape* `(free i)`*);*
- $\xi(\Gamma \vdash_{\Sigma} \langle$`x,M,N`$\rangle$`: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 6 (Adequacy of definitional equality). *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 bindable variables in* $\mathcal{X}$ *in the CBV λ-calculus and proof terms* `h` *such that* $\Gamma \vdash_{\Sigma_{CBV}}$ `h` $:$ `(eq` $\epsilon_{\mathcal{X}}(M)\ \epsilon_{\mathcal{X}}(N)$`)` *in η-long normal form, where* Γ *contains the encoding of the variables in* $\mathcal{X}$*.*

4.2 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]).

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.

$\mathsf{A}_1 : \phi \to (\psi \to \phi)$
$\mathsf{A}_2 : (\phi \to (\psi \to \xi)) \to (\phi \to \psi) \to (\phi \to \xi)$
$\mathsf{A}_3 : (\neg\phi \to \neg\psi) \to ((\neg\phi \to \psi) \to \phi)$
$\mathsf{MP} : \dfrac{\phi \quad \phi \to \psi}{\psi}$
$\mathsf{K} : \Box(\phi \to \psi) \to (\Box\phi \to \Box\psi)$
$\mathsf{4} : \Box\phi \to \Box\Box\phi$
$\top : \Box\phi \to \phi$
$\mathsf{NEC} : \dfrac{\phi}{\Box\phi}$

Fig. 7. Hilbert style rules for Modal Logic S_4

By way of example, we give the signature for classical S_4 (see Figure 8) in Hilbert style (see Figure 7), which features necessitation (rule NEC in Figure 7) as a rule of proof. Due to lack of space, we limit the encoding in Figure 8 to the most significant cases. We make use of the predicate ***Closed*** $(\Gamma \vdash_{\Sigma}$ `m:`***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[4] (see Appendix 7.5 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 $\mathtt{True}(\Box\phi)$ in the conclusion of `NEC` (see Figure 8). Adequacy theorems are rather trivial to state and prove, given the canonical version of $\mathsf{LF}_{\mathcal{P}}$.

$\mathtt{o} : \mathtt{Type} \qquad \to : \mathtt{o}^3 \qquad \neg : \mathtt{o}^2 \qquad \Box : \mathtt{o}^2 \qquad \mathtt{True} : \mathtt{o} \to \mathtt{Type}$

$\mathtt{A1} : \Pi\phi,\psi{:}\mathtt{o}.\ \mathtt{True}(\phi\to(\psi\to\phi)) \qquad \mathtt{K} : \Pi\phi,\psi{:}\mathtt{o}.\ \mathtt{True}(\Box(\phi\to\psi)\to(\Box\phi\to\Box\psi))$

$\mathtt{MP} : \Pi\phi{:}\mathtt{o}.\ \Pi\psi{:}\mathtt{o}.\ \mathtt{True}(\phi) \to \mathtt{True}(\phi\to\psi) \to \mathtt{True}(\psi)$

$\mathtt{NEC} : \Pi\phi{:}\mathtt{o}.\ \Pi\mathtt{m}{:}\mathtt{True}(\phi).\ \mathcal{L}_{\mathtt{m}}^{\mathit{Closed}}[\mathtt{True}(\Box\phi)]$

Fig. 8. The signature Σ for classic S_4 Modal Logic in Hilbert style

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 9 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 10 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 $\mathit{Boxed}(\Gamma \vdash_{\Sigma} \mathtt{m}{:}\ \mathtt{True}(\phi))$ holds in the case of S_4 iff the occurrences of free variables of $\mathtt{m}$ occur in subterms whose type has the shape $\mathtt{True}(\Box\psi)$ or is $\mathtt{o}$. In the case of S_5 the predicate holds iff the variables of $\mathtt{m}$ have type $\mathtt{True}(\Box\psi)$ or $\mathtt{True}(\neg\Box\psi)$ or occur in subterms whose type is $\mathtt{o}$. 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 $\mathtt{True}(\Box\phi)$ in the conclusion of the introduction rule `BoxI` (see Figure 10).

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

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

Fig. 9. Some Modal Logic rules (common and $S_{4,5}$ rules) in Natural Deduction style

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

[4] Another way of specifying such a property is to require that "all free variables occurring in $\mathtt{m}$ have a simple type over ι and o".

$\texttt{o:Type}\quad \texttt{and:o}^3\quad \texttt{or:o}^3\quad \texttt{→:o}^3\quad \texttt{¬:o}^2\quad \texttt{□:o}^2\quad \texttt{True:o -> Type}$

$\texttt{AndI} : \Pi\phi,\psi\texttt{:o. True(}\phi\texttt{) -> True(}\psi\texttt{) -> True(}\phi\ \texttt{and}\ \psi\texttt{)}$

$\texttt{OrE} : \Pi\phi,\psi,\xi\texttt{:o.True(}\phi\ \texttt{or}\ \psi\texttt{)->(True(}\phi\texttt{)->True(}\xi\texttt{))}$
$\texttt{->(True(}\psi\texttt{)->True(}\xi\texttt{))->True(}\xi\texttt{)}$

$\texttt{RAA} : \Pi\phi\texttt{:o. (True(}\neg\phi\texttt{) -> True(}\phi\texttt{)) -> True(}\phi\texttt{)}$

$\texttt{BoxI} : \Pi\phi\texttt{:o. }\Pi\texttt{m:True(}\phi\texttt{). }\mathcal{L}^{Boxed}_{\texttt{m}}[\texttt{True(}\Box\phi\texttt{)}]$

$\texttt{BoxE} : \Pi\phi\texttt{:o. }\Pi\texttt{m:True(}\Box\phi\texttt{). True(}\phi\texttt{)}$

Fig. 10. The signature Σ_S for classic S_4 Modal Logic in $\mathsf{LF}_{\mathcal{P}}$

Non-commutative linear logic. In this section we outline an encoding in $\mathsf{LF}_{\mathcal{P}}$ of a substructural logic like the one presented in [19]. Take, for instance, the rules for the *ordered variables* and the $\twoheadrightarrow$ introduction/elimination rules:

$$\frac{}{\star;\cdot;z{:}A\vdash z{:}A}\ ovar$$

$$\frac{\star;\Delta;(\Omega,z{:}A)\vdash M{:}B}{\star;\Delta;\Omega\vdash\lambda^{>}z{:}A.M{:}A\twoheadrightarrow B}\twoheadrightarrow I\qquad \frac{\star;\Delta_1;\Omega_1\vdash M{:}A\twoheadrightarrow B\quad \star;\Delta_2;\Omega_2\vdash N{:}A}{\star;(\Delta_1\bowtie\Delta_2);(\Omega_1,\Omega_2)\vdash M^{>}N{:}B}\twoheadrightarrow E$$

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. As was the case for the encoding of λ-calculus in the previous sections, we represent free ordered variables with constants using a coercion operator `ord:nat→term` embedding natural numbers into terms. The encodings of rules $\twoheadrightarrow I$ and $\twoheadrightarrow E$ are:

Π`M:term->term.`Π`A,B:type.`Π`z:nat.`
`(infer (ord z) A) -> (infer (M (ord z)) B) ->`
$\mathcal{L}^{Ord}_{[\texttt{(ord z)},\texttt{(M (ord z))}],\texttt{Tlist}}[$`(infer (lambdaRight M)(funRight A B))`$]$, and

Π`M,N:term.`Π`A,B:type.`
`(infer M (funRight A B) -> (infer N A) -> infer (appRight M N) B),`

where `infer:term->type->`Type is the judgment representing typed derivations in the system, `lambdaRight:(term`^{2}`)->term` and `funRight:type`^{3} are, respectively, the term constructor and the type constructor of *right ordered functions*, and $\mathit{Ord}(\Gamma\vdash_{\Sigma}$ `[(ord z),(M (ord z))]`:`Tlist`$)$ is the predicate checking that `M` and `z` are closed, and that `(ord z)` occurs only once and as the rightmost[5] constant in `(M (ord z))`.

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.

[5] The pseudocode of the predicate checking this condition can be easily adapted from that of the *freeVarOcc* and *rightmost* functions appearing in Appendix 7.4.

5 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 Appendix B, 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 $\dfrac{A \to B \quad A \equiv C \quad C}{B}$ in *Deduction Modulo* can be represented as: $\supseteq_{\equiv}$: $\Pi A, B, C{:}o.\Pi x : True(A \to B).\Pi y : True(C).\mathcal{L}^{\equiv}_{A,C}[True(B)]$.

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* ([16]).

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.
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.P. Barendregt and E. Barendsen. Autarkic computations in formal proofs. *Journal of Automated Reasoning*, 28:321–336, 2002.
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.
9. G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31:33–72, 2003.
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 and D. Licata. Mechanizing metatheory in a logical framework. *J. Funct. Program.*, 17:613–673, 2007.
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. D. Licata and R. Harper. A universe of binding and computation. In *ICFP '09*, pages 123–134. ACM, 2009.
16. A. Nanevski, F. Pfenning, and B. Pientka. Contextual Model Type Theory. *ACM Transactions on Computational Logic*, 9(3), 2008.
17. F. Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *LICS'93*, pages 221–230, 1993.
18. B. Pientka and J. Dunfield. Programming with proofs and explicit contexts. In *PPDP'08*, pages 163–173. ACM, 2008.
19. J. Polakow and F. Pfenning. Natural deduction for intuitionistic non-commutative linear logic. In *TLCA'99*, volume 1581 of *LNCS*, pages 644–644, 1999.
20. K. Watkins, I. Cervesato, F. Pfenning, and D. Walker. A Concurrent Logical Framework I: Judgments and Properties. Tech. Rep. CMU-CS-02-101, 2002.