LFP – A Logical Framework with External Predicates
Furio Honsell, Marina Lenisa, Luigi Liquori, Petar Maksimovic, Ivan Scagnetto

To cite this version:

HAL Id: hal-01146691
https://inria.hal.science/hal-01146691
Submitted on 13 May 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
LFp – 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  
Institut National de Recherche en Informatique et en Automatique, 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 LFp 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 o-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 LFp, one can factor out the complexity of encoding specific features of logical systems which would otherwise be awkwardly encoded in LF, e.g. side-conditions in the application of rules in Modal Logics, and substructural rules, as in non-commutative Linear Logic. The idea of LFp is that these conditions need only to be specified, while their verification can be delegated to an external proof engine, in the style of the Poincaré Principle. We investigate and characterize the metatheoretical properties of the calculus underpinning LFp: strong normalization, confluence, and subject reduction. This latter property holds under the assumption that the predicates are well-behaved, i.e. closed under weakening, permutation, substitution, and 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 o-modality constructors for building types of the shape \( L_{N,\sigma}[\rho] \), where \( \mathcal{P} \) is a predicate on typed 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 \( \mathcal{L}_{N,\sigma}[M] \) of type \( \mathcal{L}_{N,\sigma}[\rho] \), via the introduction rule \((I)\), presented below. Correspondingly, we introduce an unlock destructor, \( \mathcal{U}_{N,\sigma}[M] \), and an elimination rule \((E)\) which allows for the elimination of the lock type constructor, under the condition that a specific predicate \( \mathcal{P} \) is verified, possibly externally, on an appropriate correct, i.e. derivable, judgement.

\[
\begin{align*}
\Gamma \vdash_{\Sigma} M : \rho & \quad \Gamma \vdash_{\Sigma} N : \sigma \\
\Gamma \vdash_{\Sigma} \mathcal{L}_{N,\sigma}[M] : \mathcal{L}_{N,\sigma}[\rho] \quad (I) \\
\Gamma \vdash_{\Sigma} M : \mathcal{L}_{N,\sigma}[\rho] & \quad \Gamma \vdash_{\Sigma} N : \sigma \quad \mathcal{P}(\Gamma \vdash_{\Sigma} N : \sigma) \\
\Gamma \vdash_{\Sigma} \mathcal{U}_{N,\sigma}[M] & : \rho \quad (E)
\end{align*}
\]

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

LFp 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 LFp predicates is not specified, 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, LFp 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 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 LFp. 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 \( \beta \)\( \eta \)-reduction in the arguments. \( \text{LF}_P \) is decidable, if the external predicates are decidable.

Moreover, we sketch a library of external predicates, which we use to present substantial examples of encodings in \( \text{LF}_P \), that are awkward in LF. In particular, we encode the call-by-value \( \lambda \)\( \mu \) calculus, and we provide 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]. \( \text{LF}_P \) can naturally support program correctness systems and Hoare-like logics, see [15] for more details.

As far as expressivity is concerned, \( \text{LF}_P \) 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 \( \text{LF}_P \) 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 \( \text{LF}_P \). \( \text{LF}_P \) can also be viewed as a neat methodology for separating the logical contents from the verification of structural and syntactical properties, which is often cumbersome, but ultimately computable.

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 \( \beta \)-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 \emph{conditioned} \( \beta \)-reduction, which fires when the predicate holds on a \{term, judgement\}. In \( \text{LF}_P \), predicates are external to the system and the verification of the validity of the predicate is part of the typing system. Standard \( \beta \)-reduction is recovered and combined with an \emph{unconditioned} lock reduction. The move of having predicates as new type constructors rather than as parameters of \( \Pi \)'s and \( \lambda \)'s allows \( \text{LF}_P \) to be a mere language extension of standard LF. This simplifies the meta-theory, 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 relationship with the systems of [5, 7, 13, 14], which combine derivation and computation, has been discussed above. Systems supporting the Poincaré Principle [4], or Deduction Modulo [9], where derivation is separated from verification, can be directly incorporated in \( \text{LF}_P \). 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 \( \text{LF}_P \), 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, whereas in \( \text{LF}_P \) it is integrated in the system. This integration is closer to the approach of [16], but additional work is required in order to be able to precisely compare their 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, in this case as well, the focus is placed on the computational level inside the framework, rather than on the capability of delegating the verification of predicates to an external oracle.

\( \text{LF} \) with Side Conditions (LFSC), presented in [23], is more reminiscent of our approach as “it extends LF to allow side conditions to be expressed using a simple first-order functional programming language”. Indeed, the author aims at factoring the verifications of (complicated) side-conditions out of the main proof. Such a 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 \( \text{LF}_P \), the typing system, and the \( \beta \)\( \eta \)-reduction, together with the main meta-theoretical properties of the system. In Section 3, we show how to encode the call-by-value \( \lambda \)\( \mu \)-calculus, Modal Logics, and non-commutative Linear Logic. Conclusions and future work appear in Section 4. An extended version of the present paper, including a canonical version of \( \text{LF}_P \) and more examples, appears in [15].

2. The Framework

The pseudo-syntax of \( \text{LF}_P \) is presented in Figure 1 on page 2. It is essentially that of LF, with the addition, on families and objects, of a lock constructor, \( \mathcal{L}_{N,\sigma}[-] \), and a corresponding lock destructor, \( \mathcal{U}_{N,\sigma}[-] \), 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 \( \Gamma \vdash N : \sigma \). \( \text{LF}_P \) 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. For notational completeness, the list of external predicates should appear in the signature. We omit it to increase readability.

\[
\begin{align*}
\Sigma &\in S & \Sigma ::= &\emptyset | \Sigma, \alpha : K | \Sigma, \sigma : \tau & & \text{Signatures} \\
\Gamma &\in C & \Gamma ::= &\emptyset | \Gamma, x : \sigma & & \text{Contexts} \\
K &\in K & K ::= &\text{Type} | \Pi x : \sigma. K & & \text{Kinds} \\
\sigma, \tau, \rho &\in F & \sigma ::= &a | \Pi x : \sigma. \tau | \sigma N | \mathcal{L}_{N,\sigma}[\rho] & & \text{Families} \\
M, N &\in O & M ::= &e | x | \lambda x : \sigma. M | M N | \mathcal{L}_{N,\sigma}[M] | \mathcal{U}_{N,\sigma}[M] & & \text{Objects}
\end{align*}
\]

Figure 1. \( \text{LF}_P \) Syntax

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 \( \text{Dom}(\Gamma) \) 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 \( \mathcal{L}_{N,\sigma}[T] \) affects \( T, N, \) and \( \sigma \), i.e. \( \mathcal{L}_{N,\sigma}[T][M/x] = \mathcal{L}_{N,\sigma}[M/x][T] \), 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 \( \Gamma \) or \( \sigma \). We will work modulo \( \alpha \)-conversion and Barendregt’s hygiene condition. All of the symbols can appear indexed.
The type system for LF_p proves judgements of the shape:

\[
\begin{align*}
\Sigma &\vdash \text{sig} & \Sigma \text{ is a valid signature} \\
\Gamma &\vdash \Delta & \Gamma \text{ is a valid context in } \Sigma \\
\Gamma &\vdash K & K \text{ is a kind in } \Gamma \text{ and } \Sigma \\
\Gamma &\vdash \sigma : K & \sigma \text{ has kind } K \text{ in } \Gamma \text{ and } \Sigma \\
\Gamma &\vdash M : \sigma & M \text{ has type } \sigma \text{ in } \Gamma \text{ and } \Sigma
\end{align*}
\]

We denote by \( \Gamma \vdash \alpha \) any typing judgement \( \Gamma \vdash T : T' \) or \( \Gamma \vdash T \). In the two latter judgements, \( T \) will be referred to as the subject of that judgement. The typing rules of LF_p are presented in Figure 2. The rule \((F\text{-Lock})\) is used to form a lock type; the rule \((O\text{-Lock})\) is the corresponding introduction rule for building objects of the lock type, while the rule \((O\text{-Unlock})\) is the elimination rule. It applies only when the predicate \( \mathcal{P} \) holds.

In LF_p, we will have two types of reduction: standard \(\beta\)-reduction and \(\mathcal{L}\)-reduction. The latter allows for the dissolution of a lock, in the presence of an unlock (see Figure 3 for the main \(\beta\mathcal{L}\)-reduction rules on "raw terms", and Figures 4 and 5 for the contextual closure and \(\beta\mathcal{L}\)-equivalence on families, with the corresponding rules for kinds and objects handled similarly).

**Figure 3.** Main one-step-\(\beta\mathcal{L}\)-reduction rules in LF_p

In [15] we also provide a canonical presentation of LF_p, in the style of [10, 24]. Here, we present the main properties of LF_p. Without any additional assumptions concerning predicates, the type system is strongly normalizing and confluent. The former follows from strong normalization of LF (see [11]), while the latter follows from strong normalization and local confluence, using Newman’s Lemma. The proof of Subject Reduction, however, requires certain conditions to be placed on the predicates, and these conditions are summarized in the following 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 \( \mathcal{P}_i \) in this set satisfies the following conditions:

**Closure under signature, context weakening and permutation.** If \( \Sigma \) and \( \Omega \) are valid signatures with every occurrence in \( \Omega \) and \( \Gamma \) and \( \Delta \) are valid contexts with every declaration in \( \Gamma \) also occurring in \( \Delta \), and \( \mathcal{P}(\Gamma \vdash \alpha \) holds, then \( \mathcal{P}(\Delta \vdash \alpha) \) also holds.

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

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

**Strong normalization.** In order to prove strong normalization of LF_p, we will rely on the strong normalization of LF, as proven in [12]. First, we will introduce the function \( -ul \) : LF_p \rightarrow LF, which maps LF_p terms into LF terms, by deleting the \( \mathcal{L} \) and \( \mathcal{U} \) symbols from an LF_p term, while preserving all of the relevant information, in the following manner:

\[
\begin{align*}
([\text{Type}[a|x]]) &\rightarrow_{ul} ([\text{Type}[a|x]]) \\
([\Pi[\lambda]x:\sigma. T]) &\rightarrow_{ul} ([\Pi[\lambda]x:\sigma. T]) \\
(T M) &\rightarrow_{ul} T \rightarrow_{ul} M \\
([\mathcal{L}[\lambda]]_{N}[a]) &\rightarrow_{ul} (\lambda x_{f} : \sigma \rightarrow_{ul} T) N \rightarrow_{ul}
\end{align*}
\]

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 LF_p, using \( -ul \), we still translate LF_p-terms to full-fledged LF-terms, including those with abstractions in families. This is required so that the \( N \) and \( \sigma \), which index the \( \mathcal{L} \) and \( \mathcal{U} \) symbols, are not lost. We can naturally extend \( -ul \) to signatures and contexts of LF_p, obtaining signatures and contexts of LF, and then to judgements of LF_p, obtaining judgements of LF. With \( -ul \) defined in this way, using structural induction, we obtain the following propositions:

**Proposition 1.** If \( T =_{\beta\mathcal{L}} T' \) in LF_p, then \( T \rightarrow_{ul} T' \rightarrow_{ul} T'' \) in LF.

**Proposition 2.** The function \( -ul \) maps derivable judgements of LF_p into derivable judgements of LF.

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

**Proposition 3.** Only finitely many \( \beta \)-reductions can occur within any LF_p-term.

Next, we notice that any LF_p-term has only finitely many \( \mathcal{L} \)-redexes before any reductions take place, and that this number can be increased only through \( \beta \)-reductions, and only by a finite amount per \( \beta \)-reduction. However, if we were to have an LF_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 \( \beta \)-reductions in this sequence has to be finite. On the other hand, with the number of \( \beta \)-reductions in the sequence being finite, it would not be possible to reach infinitely many \( \mathcal{L} \)-redexes, and such a term \( T \) cannot exist in LF_p. Therefore, we have the Strong Normalization theorem:

**Theorem 1** (Strong normalization of LF_p).

1. If \( \Gamma \vdash \Sigma K \), then \( K \) is \( \beta \mathcal{L} \)-strongly normalizing.
2. If \( \Gamma \vdash \Sigma \sigma : K \), then \( \sigma \) 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, the proof of which is routine:

**Lemma 1** (Substitution lemma for local confluence).

1. If \( N \rightarrow_{\beta\mathcal{L}} N' \), then \( M[N/x] \rightarrow_{\beta\mathcal{L}} M'[N'/x] \).
2. If \( M \rightarrow_{\beta\mathcal{L}} M' \), then \( M[N/x] \rightarrow_{\beta\mathcal{L}} M'[N/x] \).

Next, we proceed to prove local confluence:

**Lemma 2** (Local confluence of LF_p). \( \beta\mathcal{L} \)-reduction is locally confluent, i.e., if \( T \rightarrow_{\beta\mathcal{L}} T' \) and \( T \rightarrow_{\beta\mathcal{L}} T'' \), then there exists a \( T''' \) such that \( T' \rightarrow_{\beta\mathcal{L}} T''' \) and \( T'' \rightarrow_{\beta\mathcal{L}} T''' \).

**Proof.** By simultaneous induction on the two derivations \( T \rightarrow_{\beta\mathcal{L}} T' \) and \( T \rightarrow_{\beta\mathcal{L}} T'' \). All of the cases where \( T \) is a kind or family, as well as most of the cases where \( T \) is an object are trivially proven, using the induction hypotheses. Here, we will show only the illustrative cases, involving base reduction rules:

1. Let us have, by \( (\beta\text{-Main}) \), that \( (\lambda x: \sigma. M) N \rightarrow_{\beta\mathcal{L}} (\lambda x: \sigma'. M) N \), from
Let us have that \((\beta:\text{Main})\), \((\lambda x : \sigma . M) N \rightarrow_{\beta \mathcal{L}} M[N/x]\). Let us also have that \((\lambda x : \sigma . M) N \rightarrow_{\beta \mathcal{L}} (\lambda x : \sigma . M') N\), from \(M \rightarrow_{\beta \mathcal{L}} M'\), by the object rules for closure under context. In this case, we will show that the required conditions are met for \(M''' \equiv M'[N/x]\). Indeed, by the definition of \(\rightarrow_{\beta \mathcal{L}}\), we have that \(M[N/x] \rightarrow_{\beta \mathcal{L}} M[N/x]\), and also, by the reduction rule \((\beta:\text{Main})\), we have that \((\lambda x : \sigma' . M) N \rightarrow_{\beta \mathcal{L}} M[N/x]\), effectively having \((\lambda x : \sigma' . M) N \rightarrow_{\beta \mathcal{L}} M'[N/x]\).
while we obtain that $M[N/x] \rightarrow_{\beta\ell} M'[N/x]$ from part 2 of Lemma 1.

3. Let us have, by $(\mathcal{LMain}, U_{\mathcal{L}}^0, \mathcal{L}_{\mathcal{L}}^0[M]) \rightarrow_{\beta\ell} M$, and, also, that $U_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[M]] \rightarrow_{\beta\ell} U_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[M]]$, from $N \rightarrow_{\beta\ell} N'$, by the object rules for closure under context. In this case, we will show that the required conditions are met for $M'' \equiv M$. By the definition of $\rightarrow_{\beta\ell}$, we have that $M \rightarrow_{\beta\ell} M$, which leaves us with needing to show that $U_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[M]] \rightarrow_{\beta\ell} U_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[M]]$, from which we finally obtain that $U_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[M]] \rightarrow_{\beta\ell} M$, by the reduction rule $(\mathcal{LMain})$, effectively having $U_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[M]] \rightarrow_{\beta\ell} M$. The remaining subcases are handled very similarly.

Having proven local confluence, finally, from Theorem 1, the following property follows directly from the typing and that

\[ \Gamma \vdash \tau \Rightarrow \Gamma \vdash \tau \]

implies

\[ \Gamma \vdash \tau \]

leaves us with needing to show that $U_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[M]] \rightarrow_{\beta\ell} U_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[M]]$, from which we finally obtain that $U_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[M]] \rightarrow_{\beta\ell} M$, by the reduction rule $(\mathcal{LMain})$, effectively having $U_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[M]] \rightarrow_{\beta\ell} M$. The remaining subcases are handled very similarly.

\[ \Box \]

Theorem 2 (Confluence of LF\(_P\)). \(\beta\ell\)-reduction is confluent, i.e. if $T \rightarrow_{\beta\ell} T'$ and $T \rightarrow_{\beta\ell} T''$, then there exists a $T'''$, such that $T' \rightarrow_{\beta\ell} T'''$ and $T'' \rightarrow_{\beta\ell} T'''$.

Subject reduction. We begin by proving several auxiliary lemmas and propositions:

Lemma 3 (Auxiliary properties).

1. If $\Pi \vdash \sigma, \tau \rightarrow_{\beta\ell} \tau'$, then $\Pi \vdash \sigma, \tau',$ such that $\sigma' \equiv \beta\ell \sigma$, and $\tau' \equiv \beta\ell \tau$.

2. If $\mathcal{L}_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[p]] \rightarrow_{\beta\ell} \mathcal{L}_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[p]'$, for some $\mathcal{N}'$, $\sigma'$, and $\rho'$, such that $N' \equiv \beta\ell N$, $\sigma' \equiv \beta\ell \sigma$, and $\rho' \equiv \beta\ell \rho$.

3. If $\Gamma \vdash \sigma \rightarrow_{\beta\ell} \sigma$ and $\Pi \vdash \sigma \rightarrow_{\beta\ell} \sigma'$, then $\Gamma \vdash \sigma \rightarrow_{\beta\ell} \sigma'$.

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

Proposition 4 (Abstraction typing). If $\Gamma \vdash \lambda x : \sigma . M : \Pi \vdash \sigma, \tau$, then $\Gamma \vdash \lambda x : \sigma . M : \tau$.

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

Proposition 5 (Subderivation, part 1).

1. A derivation of $\vdash \emptyset$ has a subderivation of $\Sigma$ sig.

2. A derivation of $\Sigma, \alpha \vdash \sigma$ has subderivations of $\Sigma$ sig and $\vdash \Sigma, \alpha$.

3. A derivation of $\Sigma, \alpha \vdash \sigma$ has subderivations of $\Sigma$ sig and $\vdash \Sigma, \alpha$.

4. A derivation of $\vdash \Sigma, \alpha$ has subderivations of $\Sigma$ sig and $\vdash \Sigma, \alpha$.

5. A derivation of $\vdash \Sigma, \alpha$ has subderivations of $\Sigma$ sig and $\vdash \Sigma, \alpha$.

6. Given a derivation $D$ of the judgement $\vdash \Sigma, \alpha$ 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 $\Sigma$ and $\Omega$ are valid signatures, and every declaration occurring in $\Sigma$ also occurs in $\Omega$, then $\vdash \Sigma, \alpha$ implies $\vdash \Omega, \alpha$.

2. If $\Gamma$ and $\Delta$ are valid contexts w.r.t. the signature $\Sigma$, and every declaration occurring in $\Delta$ also occurs in $\Gamma$, then $\vdash \Sigma, \alpha$ implies $\vdash \Delta, \alpha$.

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

1. If $\vdash \Sigma, \alpha \vdash K$, then $\vdash \Sigma, K$.

2. If $\vdash \Sigma, K$, then $\vdash \Sigma, \alpha$.

Proposition 8 (Transitivity). If predicates are closed under signature/context weakening and permutation and under substitution, then: if $\Gamma, \sigma \vdash \alpha$, and $\Gamma \vdash \beta : \sigma$, then $\Gamma, \beta \vdash \psi : \sigma$.

Notice that, contrary to what happens in traditional type systems, the following closure under expansion does not hold: $\Gamma \vdash \Sigma M[N/x] : \tau \Rightarrow \Gamma \vdash \Sigma (\lambda x : \sigma . M) N : \tau$, for $\Gamma \vdash \Sigma N : \sigma$.

Proposition 9 (Unicity of types and kinds). If predicates are closed under signature/context weakening and permutation and under substitution, then: if $\Gamma \vdash T : T_1$ and $\Gamma \vdash T : T_2$, then $T_1 \equiv \beta\ell T_2$.

Finally, we have Subject Reduction:

Theorem 3 (Subject reduction of LF\(_P\)). If predicates are well-behaved, then:

1. If $\Gamma \vdash \Sigma K$, and $K \rightarrow_{\beta\ell} K'$, then $\Gamma \vdash \Sigma K'$.

2. If $\Gamma \vdash \Sigma \vdash \sigma : K$, and $\sigma \rightarrow_{\beta\ell} \sigma'$, then $\Gamma \vdash \Sigma \sigma' : K$.

3. If $\Gamma \vdash \Sigma M : \sigma$, and $\sigma \rightarrow_{\beta\ell} M'$, 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 $(\text{F\text{-}Lock})$, $(\text{O\text{-}Lock})$, and $(\text{O\text{-Unlock})}$ all have an additional premise $\Gamma \vdash \sigma$ : Type, while the rule $(\text{O\text{-Unlock})}$ also has another additional premise $\Gamma \vdash \mathcal{L}_{\mathcal{L}}^0[\mathcal{L}_{\mathcal{L}}^0[p]] : \rho$.

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

Figure 6. An extension of LF\(_P\) typing rules for Subject Reduction
have \( \Gamma \vdash \Sigma : \sigma \), we obtain that \( \Gamma \vdash \Sigma \mathcal{L}^{\mathcal{P}}_{\mathcal{N},\sigma}[\rho] : \text{Type} \). From this, given \( \sigma = \rho \mathcal{L}^{\mathcal{P}}_{\mathcal{N},\sigma}[\rho] \), we obtain that \( \mathcal{L}^{\mathcal{P}}_{\mathcal{N},\sigma}[\rho] \vdash \eta \mathcal{L}^{\mathcal{P}}_{\mathcal{N},\sigma}[\rho] \), and since we already have that \( \Gamma \vdash \Sigma : \mathcal{L}^{\mathcal{P}}_{\mathcal{N},\sigma}[\rho] \), we can use the type system rule (O-Conv) to obtain \( \Gamma \vdash \Sigma : \mathcal{L}^{\mathcal{P}}_{\mathcal{N},\sigma}[\rho] \).

Finally, by the well-behavedness requirements for the predicates, we have that \( P(\Gamma \vdash \Sigma : \sigma') \) holds, and we can now use the type system rule (O-Unlock) to obtain the required \( \Gamma \vdash \Sigma : \mathcal{L}^{\mathcal{P}}_{\mathcal{N},\sigma}[\rho] : \text{Type} \). Here, we can notice that there are steps in this proof (in which we obtain \( \Gamma \vdash \Sigma : \sigma' : \text{Type} \), and \( \Gamma \vdash \Sigma : \mathcal{L}^{\mathcal{P}}_{\mathcal{N},\sigma}[\rho] : \text{Type} \)), 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 \( \mathcal{P} \) iff \( \Gamma \vdash \Sigma : \alpha \) in the original \( \mathcal{L}^{\mathcal{P}} \) system (i.e., that the judgements that these two systems derive are the same), by induction on the length of the derivation. With this, given that we have proven Subject Reduction of the extended system, we have that Subject Reduction also holds in the original \( \mathcal{L}^{\mathcal{P}} \) system.

### 3. Pragmatics and Case Studies

In this section, we illustrate the pragmatics of using \( \mathcal{L}^{\mathcal{P}} \) as a metamodel 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, or 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. The main archetype is that given constants or variables only occur with some modality \( D \) in subterms satisfying the decidable property \( C \). Modalities can include any of the phrases such as: at least once, only once, etc. \( C \) 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 \( \mathcal{L}^{\mathcal{P}} \) to access the context. As a side remark, we have noticed 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 \( \lambda \)-calculus, with a call-by-value evaluation strategy. Although this example is nicely handled using pure LF encodings already, we discuss it because it illustrates yet another way to deal with free and bound variables, using external predicates, as in the case of beta and cavi in Definition 4. Next, 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, and, due to lack of space, here provide proofs for only some of them.

**Notation:** For the sake of simplicity, in the following examples, we use the notations \( \sigma \rightarrow \tau \) for \( \Pi x : \sigma.\tau \) if \( x \notin \text{Fv}(\tau) \), and \( \sigma^{n+1} \) for the \( n \)-ary abstraction \( \sigma \rightarrow \ldots \rightarrow \sigma \). Moreover, we will omit the type \( \sigma \mathcal{L}^{\mathcal{P}}_{\mathcal{N},\sigma}[\rho] \), when \( \sigma \) is clear from the context.

In the adequacy theorems, we will use the notion of judgement in \( \eta \)-long normal form (\( \eta \)-lnf.), defined as follows:

**Definition 2** (Judgements in \( \eta \)-long normal form).

- An occurrence \( \xi \) of a constant or a variable in a term of an \( \mathcal{L}^{\mathcal{P}} \) judgement is fully applied and unlocked with respect to its type or kind \( \Pi \overrightarrow{x} : \sigma.\xi \). \( \overrightarrow{x} \) is a vector of locks, if \( \xi \) appears in contexts of the form \( \mathcal{U}_{\overrightarrow{x}}[(\ldots(\mathcal{U}_{\overrightarrow{x}}[\xi])\ldots)] \mathcal{M}_{\overrightarrow{x}} \), where \( \mathcal{M}_{\overrightarrow{x}} \) and \( \mathcal{U}_{\overrightarrow{x}} \) have the same arities of the corresponding vectors of \( \Pi \)’s and locks.

- A term \( T \) in a judgement is in \( \eta \)-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 \( \eta \)-lnf if all terms appearing in it are in \( \eta \)-lnf.

### 3.1 The untyped \( \lambda \)-calculus

#### 3.1.1 Free and bound variables.

Consider the well-known untyped \( \lambda \)-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 \( \mathcal{L}^{\mathcal{P}} \). Bindable and bound variables are modeled with variables of the metamodel, thus retaining the full Higher-Order-Abstract-Syntax (HOAS) approach, by delegating \( \alpha \)-conversion and capture-avoiding substitution to the metamodel. Such an approach allows us to abide by the “closure under substitution” condition for external predicates, while still retaining the ability to handle “open” terms explicitly.

The aforementioned “bindable” variables must neither be confused with bound, nor with free variables. For instance, the \( \lambda \)-term \( x \) (in which the variable is free) will be encoded by means of the term \( \Gamma \vdash \text{(free n): term} \) for a suitable (encoding of a) natural

1 See [15] for a presentation of \( \mathcal{L}^{\mathcal{P}} \) using canonical forms, in the style of [20, 24].
We use 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 \( \lambda \)-calculus, we put:

\[
\epsilon_X(x_i) = \begin{cases} x_i, & \text{if } x_i \in X \\ \text{free,} & \text{if } x_i \not\in X \end{cases}
\]

\[
\epsilon_X(MN) = (\text{app } \epsilon_X(M) \epsilon_X(N))
\]

\[
\epsilon_X(\lambda x.M) = (\text{lam } \lambda x.\text{term} \epsilon_X(x)(M))
\]

where, in the latter clause, \( x \not\in X \).

**Theorem 4 (Adequacy of syntax).** Let \( \{x_i\}_{i \in \mathbb{N}\setminus\{0\}} \) be an enumeration of the variables in the \( \lambda \)-calculus. Then, the encoding function \( \epsilon_X \) is a bijection between the \( \lambda \)-calculus terms with bindable variables in \( X \) and the terms \( \hat{M} \) derivable in judgments \( \Gamma \vdash_{\Sigma} \hat{M} : \text{term} \) in \( \eta \)-Inf, where \( \Gamma = [\lambda x.\text{term} | x \in X] \). Moreover, the encoding is compositional, i.e., for a term \( M \), with bindable variables in \( X = \{x_1, \ldots, x_k\} \), and \( N_1, \ldots, N_k \) with bindable variables in \( \mathcal{J} \), the following holds: \( \epsilon_X[M[x_1 / x_1, \ldots, x_k / x_k]] = \epsilon_X(M)[\epsilon_X(N_1), \ldots, \epsilon_X(N_k) / x_1, \ldots, x_k] \).

**Proof.** The injectivity of \( \epsilon_X \) follows by a straightforward inspection of its definition, while the surjectivity follows by defining the “decoding” function \( \delta_X \) on terms in \( \eta \)-Inf:

\[
\delta_X(\text{free }) = x_i \quad \text{(where } x_i \not\in X)\\
\delta_X(x_i) = x_i \quad \text{(where } x_i \in X)\\
\delta_X(\text{app } M N) = \delta_X(M) \delta_X(N)\\
\delta_X(\text{lam } \lambda x.\text{term} N) = \lambda x.\delta_X(x)(N)
\]

Given the characterization of \( \eta \)-Inf's, and the types of the constructors introduced in \( \Sigma \), it is easy to see that \( \delta_X \) is total and well-defined. It is not possible to derive a \( \eta \)-long normal form of type \( \text{term} \) containing a \( \eta \)-term, since no constructors in \( \Sigma \) use \( \eta \)-types. Finally, by induction on the structure of \( M \), it is possible to check that \( \delta_X(\epsilon_X(M)) = M \) and that \( \epsilon_X \) is compositional.

**3.1.2 The call-by-value reduction strategy.**

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

\[
\text{CBV} M = M \\
\text{CBV} N = M \\
\text{CBV} M N = M P \\
\text{CBV} M N P = M' N' \\
\text{CBV} \langle \lambda x.\text{term} \rangle v = M[v/x] \\
\text{CBV} \lambda x.\text{term} M = \lambda x.\text{term} M
\]

**Definition 4 (CBV signature \( \Sigma_{\text{CBV}} \) for \( \lambda \)-calculus CBV reduction).** We extend the signature of Definition 3 as follows:

\[
\text{triple : Type} \\
\text{triple} \to \text{term} \to \text{term} \to \text{term} \\
\text{refl : \Pi M. (eq M M)} \\
\text{symm : \Pi M N. (eq M N) \to (eq N M)} \\
\text{trans : \Pi M N P. (eq M N) \to (eq N P) \to (eq M P)} \\
\text{eq.app : \Pi M M' N'. \text{term} M N \to \text{term} M' N'} \\
\text{betav : \Pi M N. \text{term} L^\delta_{\mathcal{J}}[\text{eq (app (lam M N) (M N))}] \\
\text{csibv : \Pi M N. \text{term} L^\delta_{\mathcal{J}}[\text{eq (M x) (N x)}] \to \text{eq (lam M (lam M N))}]}
\]

where the predicates \( \mathcal{V} \) and \( \xi \) are defined as follows, and \( \text{triple} \) is the type of triples of types with types \( \text{term} \times \text{term} \times \text{term} \):

\[
- \mathcal{V} (\Gamma \vdash_{\Sigma} \text{N : term}) \text{ holds iff either } \text{N is an abstraction or } \text{N is a constant (i.e. a term of the shape (free i))} \\
- (\xi (\Gamma \vdash_{\Sigma} x, N, \text{N : triple}) \text{ holds iff } x \text{ is a constant (i.e. a term of the shape (free i)))}
\]

**Theorem 5 (Adequacy of CBV reduction).** Given an enumeration \( \{x_i\}_{i \in \mathbb{N}\setminus\{0\}} \) of the variables in the \( \lambda \)-calculus, there is a bijection between derivations of the judgment \( \Gamma \vdash_{\Sigma_{\text{CBV}}} M = N \) on terms with no bindable variables in the \( \lambda \)-calculus and proof terms \( \mathcal{H} \), such that \( \vdash_{\Sigma_{\text{CBV}}} h : \text{eq}(\mathcal{E}(M), \mathcal{E}(N)) \) is \( \eta \)-long normal form.

**Proof.** We define an encoding function \( \epsilon^\mathcal{H}_\mathcal{J} \) by induction on derivations of the form \( \text{CBV} M = N \) (on terms with no bindable variables) as follows:

\[
\begin{align*}
&\text{if } \Gamma \text{ is the derivation with only (refl), we have that } \epsilon^\mathcal{H}_\mathcal{J}(\mathcal{V}) = (\text{refl } \mathcal{E}(M))(\text{eq } \mathcal{E}(M))(\text{eq } \mathcal{E}(N)); \\
&\text{if } \Gamma \text{ is the derivation with (symm) as the last applied rule, then, by the inductive hypothesis, there is a term } h \text{ such that } \vdash_{\Sigma_{\text{CBV}}} h : \text{eq}(\mathcal{E}(N), \mathcal{E}(M)); \text{ hence, we will have that } \epsilon^\mathcal{H}_\mathcal{J}(\mathcal{V}) = (\text{symm } \mathcal{E}(M))(\mathcal{E}(N))h : (\text{eq } \mathcal{E}(M))(\mathcal{E}(N)); \\
&\text{if } \Gamma \text{ is the derivation with (trans) as the last applied rule, then, by the inductive hypothesis, we have that there exist terms } h \text{ and } h', \text{ such that } \vdash_{\Sigma_{\text{CBV}}} h : (\text{eq } \mathcal{E}(M))(\mathcal{E}(N)), \text{ and also } \vdash_{\Sigma_{\text{CBV}}} h' : (\text{eq } \mathcal{E}(N))(\text{eq } \mathcal{E}(P)); \text{ hence, we will have that } \epsilon^\mathcal{H}_\mathcal{J}(\mathcal{V}) = (\text{trans } \mathcal{E}(M))(\mathcal{E}(N))(\text{eq } \mathcal{E}(P))h : (\text{eq } \mathcal{E}(M))(\mathcal{E}(P)); \\
&\text{if } \Gamma \text{ is the derivation with (eq.app) as the last applied rule, then, by the inductive hypothesis, we have that there exist terms } h \text{ and } h' \text{ such that } \vdash_{\Sigma_{\text{CBV}}} h : (\text{eq } \mathcal{E}(M))(\mathcal{E}(N)) \text{ and } \vdash_{\Sigma_{\text{CBV}}} h' : (\text{eq } \mathcal{E}(M'))(\text{eq } \mathcal{E}(N')); \text{ hence, we will have that } \epsilon^\mathcal{H}_\mathcal{J}(\mathcal{V}) = (\text{eq.app } \mathcal{E}(M))(\mathcal{E}(N))(\text{eq } \mathcal{E}(M'))(\mathcal{E}(N'))h : (\text{eq } \mathcal{E}(M))(\mathcal{E}(N)); \\
&\text{if } \Gamma \text{ is the derivation with } \xi \text{ as the last applied rule, then, by the inductive hypothesis, there is a term } h \text{ such that } \vdash_{\Sigma_{\text{CBV}}} h : (\text{eq } \mathcal{E}(M))(\text{eq } \mathcal{E}(N))^2. \text{ Hence, we will have that } \epsilon^\mathcal{H}_\mathcal{J}(\mathcal{V}) = (\text{let.triple } \mathcal{E}(x)(\lambda x.\text{term } \mathcal{E}(x)(\lambda x.\text{term } \mathcal{E}(x)(N)) \text{app } x))
\end{align*}
\]

\( \)Notice that the object variable \( x \) occurring in \( M \) and \( N \) is represented by a constant \( (\text{free } k) \), for a 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 LF.\( _F \).
The injectivity of $\varepsilon_2^o$ follows by a straightforward inspection of its definition, while the surjectivity follows by defining the “decoding” function $\delta_0$ by induction on the derivations of the shape $\Gamma \vdash_{S_{CBV}} h : \text{term}$ in $\eta$-long normal form. Since all the cases are rather straightforward, we analyze only the definition concerning the main rule ($\beta$), since it involves an external predicate. So, if we derive from $\Sigma_{CBV}$ a proof term $h$ in $\eta$-long normal form such as $\text{betav}\ N$ whose type is $\text{eq} \ (\text{app} \ 1 \text{am}\ M)\ N$, (where $M$ denotes $\times$, with $\eta$ in $\text{Inr}$), then the predicate $\text{Val} \ ((\cdot):\Sigma_{CBV}\ N : \text{term})$ must hold, and $N$ is encoding the value $\delta_0(h)$. Hence, the decoding of $h$ is the following derivation:

$$\delta_0(h) \text{ is a value}$$

$$\Gamma \vdash_{S_{CBV}} \delta_0(\text{lam} \ (\lambda x: \text{term} \ . \ \varepsilon_2^o(M))) \delta_0(N) = \delta_0((\lambda x: \text{term}^o(M)) \ N),$$

and since we have that $\delta_0((\text{lam} \ (\lambda x: \text{term}^o(M)))) = \lambda x.\delta_2(x^o)(\text{M}(\text{N}))$ (see proof of Th. 4), that $\lambda x.\delta_2(x^o)(\text{M}(\text{N}))$ = $\lambda x.\delta_2(x^o)(\text{M}(\text{N}))$ (by induction on the structure of $N$), we are done. Therefore, it is easy to verify by induction on $\eta$-long normal forms that $\delta_0^o$ is well-defined and total. Similarly, we can prove that $\delta_0^o$ is the inverse of $\varepsilon_2^o$, making $\varepsilon_2^o$ a bijection. 

$\square$

### 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 $\text{LF}_P$ allows for smooth encodings of 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.

By way of example, we give the signature for the classical $S_4$ Modal Logic in Hilbert style (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 $\text{Closed} \ (\Gamma \vdash \Sigma : n : \text{true} \ (\phi))$, which holds iff “all free variables occurring in $n$ have type $o$”. 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 $\eta$-Infs, that there cannot be free variables of type $\forall \ (\ldots)$ in the proof term, i.e. the encoded modal formula does not depend on any assumption (see [15] for a formal specification of the predicate). This example requires that predicates inspect the environment and be defined on typed judgments, as indeed is the case in $\text{LF}_P$. The above predicate is well-behaved. As in the previous examples, we ensure a sound derivation in $\text{LF}_P$ of a proof of $\Box \phi$, by locking the type $\text{true} \ (\Box \phi)$ in the conclusion of NEC (see Figure 8).

Adequacy theorems are rather trivial to state and prove; as usual we define an encoding function $\varepsilon_X$ on formulae with free variables in $X$ as follows, representing atomic formulae by means of $\text{LF}_P$ metavariables:

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$\varepsilon_X(\phi)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi \to \psi$</td>
<td>$\text{true}(\phi \to \psi)$</td>
</tr>
<tr>
<td>$\phi \to \psi$</td>
<td>$\text{true}(\phi \to \psi)$</td>
</tr>
<tr>
<td>$\Box \phi$</td>
<td>$\text{true}(\Box \phi)$</td>
</tr>
<tr>
<td>$\Box \phi$</td>
<td>$\text{true}(\Box \phi)$</td>
</tr>
</tbody>
</table>

$\vdash X \vdash_{\Box} \text{true}(\phi)$ holds in the case of $S_4$ iff the occurrences of free variables of $\phi$ occur in subterms whose type has the shape $\text{true}(\Box \psi)$ or is $\Box$. In the case of $S_5$ the predicate holds iff the variables of $\phi$ have type $\text{true}(\Box \psi)$ or $\Box$. It is easy to
check that these predicates are well behaved. Again, the “trick” to ensure a sound derivation in LFₚ of a proof of □φ is to lock appropriately the type True(□ϕ) in the conclusion of the introduction rule BoxI (see Figure 10).

\[
\Gamma \vdash \phi, \Gamma \vdash \psi \quad (\land I)
\]

\[
\Gamma \vdash \phi \land \psi, \Gamma \vdash \xi, \Gamma, \psi \vdash \xi \quad (\land E)
\]

\[
\Delta \vdash \Box \Gamma \quad (\Box I \cdot S₄)
\]

\[
\Box \Gamma_0, \neg \Box \Gamma_1 \vdash \phi \quad (\Box I \cdot S₅)
\]

\[
\Delta \vdash \Box \phi \quad (\Box E \cdot S₄)
\]

\[
\Delta \vdash \Box \phi \quad (\Box E \cdot S₅)
\]

\[
\Gamma, \neg \phi \vdash \phi \quad (\text{RRA})
\]

Figure 9. Some Modal Logic rules (common and S₄,5 rules) in Natural Deduction style

\[
o : \text{Type} \quad \text{and} \quad o : o^3 \rightarrow o^3 \quad o : o^2
\]

True : o→Type

AndI : \Pi o, o'. True ϕ → True ψ → True (ϕ and ψ)

OrE : \Pi o1, o2. True ϕ → (True ϕ → True (ο1)) → True (True ϕ → True (ο2))

RRA : \Pi o1. (True (¬ϕ) → True (ϕ)) → True (ϕ)

BoxI : \Pi o1,o1. \Pi m : True ϕ. Lboxed [True (□ο1)]

BoxE : \Pi o1,o1. \Pi m : True (□ο1). True (ϕ)

Figure 10. The signature Σ₅ for classic S₄ Modal Logic in LFₚ

The adequacy of the truth system of S₄/S₅ can be proved by structural induction on derivations of the judgment \( \Gamma \vdash \phi \):

**Theorem 9** (Adequacy of S₄/S₅). Let \( \mathcal{X} = \{x₁, \ldots, xₙ\} \) be a set of propositional variables occurring in formulae \( \phi₁, \ldots, \phiₙ, \phi \). Then, there exists a bijection between derivations of the judgement \( \{\phi₁, \ldots, \phiₙ\} \vdash \phi \) in S₄/S₅, and proof terms \( h \) such that \( \Gamma \vdash h : \text{True} \epsilon Χ(\phi) \) in η-inf, where \( \Gamma = \{x₁: o, \ldots, xₙ: o, h₁ : \text{True} \epsilon Χ(\phi₁), \ldots, hₙ : \text{True} \epsilon Χ(\phiₙ)\} \).

### 3.2.3 Non-commutative linear logic (NCLL).

In this section, we outline an encoding in LFₚ of a substructural logic like the one presented in [21]. Take, for instance, the rules for the ordered variables and the → introduction/elimination rules:

\[
\begin{align*}
\Gamma; z: A & \vdash z: A \quad (O \cdot \text{Var}) \\
\Gamma; \Delta; \Omega & \vdash \lambda x : A. M : B \quad (\to I) \\
\Gamma; \Delta; \Omega & \vdash \lambda x : A. M : B \rightarrow B \quad (\to E)
\end{align*}
\]

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 LFₚ-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 LFₚ, introduce suitable predicates in order to enforce such constraints, without resorting to complicated encodings. In the following, we present an encoding of this ordered fragment of NCLL into LFₚ. In order to give a shallow encoding, we do not represent explicitly the proof terms of the original system (see, e.g., [21]). For an alternative deep representation, see [15]. The encodings of rules \( \to I \) and \( \to E \) are:

**impRightIntro**: \( \Pi A : o. \Pi M : \text{True } A \rightarrow \text{True } B \).

**impRight**:

\( \Pi \Pi : \text{True } A \rightarrow \text{True } B \).

**impRightE**: \( \Pi A : o. \Pi M : \text{True } A \rightarrow \text{True } B \).

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

**Theorem 8** (Adequacy of S₄/S₅ formule syntax). The encoding function \( \epsilon _X \) is a bijection between the modal logic formulæ with free variables in X and the terms \( \phi \) derivable in judgements \( \Gamma \vdash \phi : \text{in } \eta\text{-inf}, \) where \( \Gamma = \{x : o \mid x \in \mathcal{X}\} \). Moreover, the encoding is compositional, i.e. for a formula \( \phi \), with free variables in \( \mathcal{X} \), and \( \psi₁, \ldots, \psi_k \), with free variables in \( \mathcal{Y} \), the following holds: \( \epsilon _X (\phi [\psi₁, \ldots, \psi_k/x₁, \ldots, x_k]) = \epsilon _X (\phi) [\epsilon _X (\psi₁), \ldots, \epsilon _X (\psi_k)/x₁, \ldots, x_k] \).
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 LF\( _P \), 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, LF\( _P \) within LF\( _P \).

We believe that LF\( _P \) 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 simply need to extend the library of predicates.

In LF\( _P \), one can easily incorporate systems which separate derivation and computation. E.g. the rule

\[
A \to B \quad A \equiv C \quad C \quad B
\]

in Deduction Modulo can be represented as:

\[
\exists z : \Pi A, B, C : o. \\
\Pi x : \Pi (A \to B). \Pi y : \Pi (True(C)). \Pi z : \Pi (True(B)).
\]

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 \textit{a priori} ([17]).

The machinery of lock derivations is similar to \( \delta \)-rules \textit{à la} Mintschke, see [3], when we take lock rules, at object level, as \( \delta \)-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 LF\( _P \) 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 \( \lambda x : \mathbb{N}, \Pi \delta \mathcal{M} \to \mathbb{N} \cdot (\mathcal{M} \to (x - 1)) \). More generally, if we want to enforce a pre-condition \( P \) on \( M \) and a post-condition \( Q \) on the result of the computation \( FM \), we can easily express it in LF\( _P \) by means of

\[
\lambda x : \mathbb{N} \cdot (\mathcal{M} \to (x - 1)) \cdot (FM).
\]

A prototype of LF\( _P \) is currently under development. This experiment will help pick, among the many implementations of type theory in the literature, the best one with regard to proving the well-behavedness of predicates.

Acknowledgments

This work was supported by the Serbian Ministry of Education and Science projects ON174026 and II044006, and by the Italian Ministry of Education, University and Research (PRIN Project SISTER 200881XMYN and FIRB Project RBIN04M8S8).

References


