An Open Logical Framework

An Open Logical Framework*

Furio Honsell[1], Marina Lenisa[1], Luigi Liquori[2], Petar Maksimovic[2,3] and 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*

May 5, 2014

dedicated to Arnon Avron
on the occasion of his 60th birthday

Abstract

The $\mathsf{LF}_{\mathcal{P}}$ Framework is an extension of the Harper-Honsell-Plotkin's Edinburgh Logical Framework LF with *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 predicate, possibly external to the system, 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 would otherwise be awkwardly encoded in LF, *e.g.* side-conditions in the application of rules in Modal Logics, and sub-structural rules, as in *non-commutative Linear Logic*. The idea of $\mathsf{LF}_{\mathcal{P}}$ 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* or *Deduction Modulo*. Indeed such paradigms can be adequately formalized in $\mathsf{LF}_{\mathcal{P}}$. We investigate and characterize the meta-theoretical properties of the calculus underpinning $\mathsf{LF}_{\mathcal{P}}$: 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. Moreover, we provide a *canonical* presentation of $\mathsf{LF}_{\mathcal{P}}$, based on a suitable extension of the notion of $\beta\eta$-*long normal form*, allowing for smooth formulations of adequacy statements.

1 Introduction

The Edinburgh Logical Framework LF, presented in [15], is a first-order[1] constructive type theory. It was first introduced as a *general meta-language for logics*, as well as a specification language for *generic proof-checking/ proof-development environments*. In this paper, we consider an extension of LF with *external predicates*, and in this sense our framework is an *open* Logical Framework. This is accomplished by defining *lock type constructors*, which are a sort of $\diamond$-*modality constructors* for constructing types of the shape $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, where $\mathcal{P}$ is an external predicate on typed judgements.

Following the standard specification paradigm in Constructive Type Theory, we define locked types using *introduction*, *elimination*, and *equality rules*. Namely, we introduce a lock *constructor* for building objects $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]$ of type $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, via the *introduction rule* $(O{\cdot}Lock)$, presented below. Correspondingly, we introduce an unlock *destructor*, $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]$, and an *elimination rule* $(O{\cdot}Unlock)$ 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.

*This work was supported by the Serbian Ministry of Education, Science, and Technological Development (projects ON174026 and III044006).

[1] In contrast to the systems on the top and back sides of the Barendregt's λ-cube, which are either second- or higher-order.

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

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

The *equality rule* for lock types amounts to a new form of reduction we name *lock reduction* ($\mathcal{L}$-reduction), $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\mathcal{L}} M$, which allows for 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.

So as to allow the reader an introductory insight into the application and potential benefits of lock types, we present here a fragment of one of the case studies proposed later in the paper (see Section 6.3.1). The problem of suitably encoding systems with "rules of proof" (as opposed to "rules of derivation")[2] like, e.g., classical Modal Logics K, KT, $K4$, $KT4$ (S_4), $KT45$ (S_5) in Hilbert style is well-known in the literature. In this specific case, the aforementioned systems all feature necessitation as a rule of proof. We can encode such a rule in $\mathsf{LF}_{\mathcal{P}}$ by locking the type $\texttt{True}(\Box\phi)$ in the conclusion of `NEC` as follows:

```
o     :  Type
□     :  o -> o
True  :  o -> Type
NEC   :  Πφ : o. Πm : True(φ). L^Closed_m[True(□φ)]
```

where `o` denotes the type of propositions, $\Box$ is the operator commonly used to denote necessitation, and `True` is the classical truth judgment. The gist of this encoding approach is to use the predicate $\mathit{Closed}(\Gamma \vdash_\Sigma \texttt{m}{:}\mathit{True}(\phi))$ in order to correctly capture the notion of "rules of proof". This predicate holds iff "all free variables occurring in `m` have type `o`". The more detailed meaning of this predicate is further discussed in Section 6.3.1, where it is shown that it precisely captures the side-condition on the application of necessitation. The conciseness and elegance of the proposed representation in $\mathsf{LF}_{\mathcal{P}}$ can be contrasted with the burden of the extra-judgments and structures needed in other more "traditional" approaches in the literature [1, 2, 12].

$\mathsf{LF}_{\mathcal{P}}$ is parametric over a potentially unlimited 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 main idea being that their truth is to be verified via a *call* to an *external validation tool*; 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 {checking, verification, computation} of such predicates to an external proof engine or tool. The proof terms themselves do not contain explicit evidence for external predicates, but just record that a verification {has to be (lock), has been successfully (unlock)} carried out. In this manner, 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* [5].

In this paper, we develop the meta-theory of $\mathsf{LF}_{\mathcal{P}}$. Strong normalization and confluence are proven without any additional 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 [36, 16], based on a suitable extension of the notion of $\beta\eta$-long normal form. This allows for simple proofs of adequacy of the encodings.

In particular, we encode in $\mathsf{LF}_{\mathcal{P}}$ the call-by-value λ-calculus and discuss a possible extension which supports the *design-by-contract* paradigm. We provide smooth encodings of side conditions in the rules of Modal Logics, both in Hilbert and Natural Deduction styles, *cf.* [2, 12]. We also encode sub-structural logics, *i.e.* non-commutative Linear Logic, *cf.* [31, 12]. We also illustrate how $\mathsf{LF}_{\mathcal{P}}$ can naturally support *program correctness* systems and Hoare-like logics. In our encodings, we utilize a *library* of external predicates, the pseudo-code of which appears in [18].

[2] 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*.

As far as expressiveness 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 also be viewed as a neat methodology for separating the logical-deductive contents from, on one hand, the verification of structural and syntactical properties, which are often needlessly cumbersome but ultimately computable, or, on the other hand, from more general means of validation.

Synopsis. In Section 2, we present the syntax of $\mathsf{LF}_{\mathcal{P}}$, the typing system, and the $\beta\mathcal{L}$-reduction. In Section 3, we prove the main meta-theoretical properties of the system. In Section 4, the expressive power of $\mathsf{LF}_{\mathcal{P}}$ is discussed. In Section 5, we present a canonical version of $\mathsf{LF}_{\mathcal{P}}$, and we discuss the correspondence with the full framework. In Section 6, we show how to encode the call-by-value λ-calculus, a minimal functional language following the *design by contract* paradigm, Modal Logics, non-commutative Linear Logic and Hoare Logic. In Section 7, we provide one final look back on $\mathsf{LF}_{\mathcal{P}}$, while conclusions and future work appear in Section 8.

1.1 A Philosophical Prelude

Since Euclid first introduced the concept of *rigorous proof* and the *axiomatic/deductive method*, philosophers have been questioning the nature of mathematics: is it, essentially, *analytic* or *synthetic*? We shall not address these issues here, because we do not wish to dare to comment on the reflections of giants such as Leibniz, Kant and Schopenhauer. However, we humbly believe that the topics in this paper should be cast against that background, and we will, therefore, offer some comments in that direction.

Possible, but clearly partial, modern readings of the synthetic vs analytic opposition are, in our view, those of *deduction from axioms* vs. *computation according to rules*, *proof checking* vs. *verification*, and proving *inhabitability* of judgements vs. *definitional equality* of types.

The machinery of *locking/unlocking types*, which we are introducing in $\mathsf{LF}_{\mathcal{P}}$, allows for the opening up of the Logical Framework to alternate means of providing evidence for judgements. In standard LF, there are only two ways of providing evidence, namely discovering types to be inhabited or postulating that types are inhabited by introducing appropriate constants. The *lock/unlock types* of $\mathsf{LF}_{\mathcal{P}}$ allow for an intermediate level, one provided by external means, such as computation engines or automated theorem proving tools. However, among these, we could also think of graphical tools based on neural networks, or even intuitive visual arguments, as were used in ancient times for giving the first *demonstrations* of the Pythagoras' theorem, for instance. In a sense, $\mathsf{LF}_{\mathcal{P}}$, in allowing to formally accommodate any alternative proof method to pure axiomatic deduction, vindicates all of the "proof cultures" which have been utilized pragmatically in the history of mathematics, and not only in the Western tradition.

A natural objection which can be raised against $\mathsf{LF}_{\mathcal{P}}$ is: "Alternative proof methods are not rigorous enough! We need to go through the pains of rigorous formalized proof checking in order to achieve the highest reliability of our certifications!". This is, of course, true, but a few points need to be made.

First of all, absolute certainty is a myth, as it cannot be achieved. The *De Bruijn Principle* [5, 13] is usually invoked in this respect. It amounts to the request that the core of the proof checker be small and verifiable by hand. Alternate proof techniques certainly do not satisfy it in a strict sense. But alternate proof techniques, if properly recorded, are not useless and come, somewhat at an intermediate level between rigorous encoding and blatant axioms. They can expedite verifications, as in the case of the *Poincaré's Principle* [5], or *Deduction Modulo* [14], or make the proof more perspicuous and provide some intuition, as Schopenhauer advocated [33]. On a lighter note, just recall the story of the famous mathematician lecturing at the seminar, who, halfway through the proof, said: "And this trivially holds!" Just to say a few seconds later: "But is it really trivial here? Hmm...". And after about ten minutes of silence triumphantly exclaiming: "Yes, it is indeed trivial!". How should we encode such evidence? Should we just rule it out?

However, there is a far deeper reason why a fundamentalist approach to certainty cannot be maintained that easily, and this has to do with the issue of *adequacy*. Contrast, for a moment, the process of proving a computation correct w.r.t. carrying out its verification by directly executing it. Consider, for example, that $1^1 * 2^2 * 3^3 = 108$. In the latter case, one would need to do some simple arithmetics, while in the former case, one would need to reify the rules for computing exponentials and products.

Of course, using the *autarkic* approach explained in [5] or reasoning by reflection as in [9], one could internalize the needed arithmetics checking procedures (proving their correctness once and for all), while still preserving the de Bruijn principle and keeping proof terms small. However, our approach is more "schematic", in the sense that it creates room for "plugging-in" any verifier, without the need to specify which one and without the need to prove internally its soundness.

But what can guarantee that this formalization is adequate, *i.e.* that it corresponds to our intended understanding of arithmetic? The issue of proving that formal statements, such as *specifications*, *encodings*, and *proof obligations*, do indeed correspond to the intended meanings and pragmatic usages cannot ever be done completely internally to any system. Ultimately, we have to resort to some informal argument outside any possible De Bruijn Principle. And any such argument can, at best, increase our confidence in the correctness of our proof of the arithmetical computation. If one looks for a definitive proof of adequacy, one is led into an infinite regress. The moral here is that only fully internalized arguments can rely on the De Bruijn Principle, but even the simplest application takes us outside the system. Ultimately, we have to "just do it" as in Munchhausen trilemma, or as in the story by Lewis Carroll on the dialogue between "A-kill-ease" and the "Taught-us" [8].

One final comment on lock/unlock types vs. Deduction Modulo or the Poincare's Principle, which will be slightly expanded in Section 7. The latter are always extensions of the type Equality Rule to new definitional equalities. $\mathsf{LF}_{\mathcal{P}}$ on the other hand, permits a reflection into the proof objects themselves.

One concluding comment. The traditional LF answer to the question "What is a Logic?" was: "A signature in LF". In $\mathsf{LF}_{\mathcal{P}}$, we can give the homologue answer, namely "A signature in $\mathsf{LF}_{\mathcal{P}}$", since external predicates can be read off the types occurring in the signatures themselves. But we can also use this very definition to answer a far more intriguing question: "What is a Proof culture?".

1.2 Comparison with Related Work

The present paper extends [21], and continues the research line of [17, 19], which present extensions of the original LF, where a notion of β-reduction *modulo* a predicate $\mathcal{P}$ is considered. These are based on the idea of *stuck-reductions* in objects and types in the setting of higher-order term rewriting systems, by Cirstea-Kirchner-Liquori [10], later generalized to a framework of Pure Type Systems with Patterns [6]. This typing protocol was essential for the preservation of strong normalization of typable terms, as proven in [17]. In [17, 19] 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 parameters of Π's and λ's allows $\mathsf{LF}_{\mathcal{P}}$ to be a mere *language extension* of standard LF. This simplifies the meta-theory, and provides a more modular approach.

Our approach generalizes and subsumes, in an abstract way, other approaches in the literature which combine internal and external derivations. In many cases, it can express and incorporate these approaches. The relationship with the systems of [10, 6, 17, 19], which combine derivation and computation, has been discussed above. Systems supporting the *Poincaré Principle* [5], or *Deduction Modulo* [14], where derivation is separated from verification, can be directly incorporated in $\mathsf{LF}_{\mathcal{P}}$. Similarly, we can abstractly subsume the system presented in [7], 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. Another system which has a very similar goal w.r.t. $\mathsf{LF}_{\mathcal{P}}$ is presented in [11], where a framework named $\lambda\Pi$-calculus modulo is introduced, extending the original LF with computation rules. The latter are realized by means of rewrite rules empowering the "traditional" conversion rule of LF (*i.e.*, the congruence relation $\equiv_\beta$ is replaced by $\equiv_{\beta\mathcal{R}}$, where $\mathcal{R}$ denotes the set of rewrite rules introduced into the system). The authors then successfully encode all functional Pure Type Systems (PTS) into the $\lambda\Pi$-calculus modulo, proving the conservativity of their embedding under the termination hypothesis. The main difference between $\lambda\Pi$-calculus modulo and $\mathsf{LF}_{\mathcal{P}}$ amounts to the fact that the latter features a simpler metatheory, because the reduction is closer to the standard β-reduction (at least in principle) and the external predicates are handled in a more controlled way by means of the lock/unlock mechanism. The direct consequence of this approach, from a practical point of view (when considering a possible implementation), is that we do not need to change the kernel of the original LF, but only *extend* it.

In [35], an extension of the Edinburgh LF with an equational theory is proposed, opening the door to new ways of conversions among types within the framework. As a consequence, strong normalization and confluence properties remain valid only in a weaker form (namely, modulo the equivalence induced by the equational theory on types). In the second part of the work, Higher-order Term Rewriting Systems (HTRS) with dependent types are introduced and used to generate equational theories, much like those analyzed in the first part. Of course, the rewriting rules of such an HTRS must adhere to some constraints, in order to guarantee the fundamental properties of the extended LF. For instance, it is forbidden to use a rewrite rule to rewrite the type of another rule, *i.e.*, rewriting must preserve well-typedness of expressions. According to the author, the benefit of the new system, w.r.t. the original LF, is to overcome the inadequacies emerging when dealing with the encoding of object languages embodying notions of computations via equational rules. This work has served as a stepping stone to constraint-based extensions of the proof assistant Twelf, which are called *constraint domains* [26]. These extensions provide a way for users to work easily with objects (such as rational numbers), the explicit formalization of which in Twelf would otherwise be quite lengthy or inefficient, but are still considered to be highly experimental. As future work, it would be interesting to study the possibility of "embedding" the HTRSs, as in [35], as well as the constraint domains of Twelf in $\mathsf{LF}_{\mathcal{P}}$, as external predicates, since the constraints imposed on the rewriting rules seem closely related to our well-behavedness properties (see Definition 1). Indeed, just as in [35], we have also used Newman's Lemma (see Section 3) to prove confluence of our system, and the issue of preserving well-typedness of expressions has also been our main concern throughout the development of the meta-theory of $\mathsf{LF}_{\mathcal{P}}$.

The work presented here also has a bearing on proof irrelevance. In [25], 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, and the *irrelevant* terms need to be derived in the system anyway. With our approach, we move one step further, and do away completely with *irrelevant* terms in the system by simply delegating the task of building them to the external proof verifier. In $\mathsf{LF}_{\mathcal{P}}$, we limit ourselves to the recording, through a lock type, that one such evidence, possibly established elsewhere, 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 [27, 28], although in that approach the inspection was layered upon LF, whereas in $\mathsf{LF}_{\mathcal{P}}$ it is integrated in the system. This integration is closer to the approach of [22], but additional work is required in order to be able to compare their expressive powers precisely.

Another interesting framework, which adds a layer on top of LF is the Delphin system [32], 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.

LF with Side Conditions (LFSC), presented in [34], 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).

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

The pseudo-syntax of the $\mathsf{LF}_{\mathcal{P}}$ system is presented in Figure 1. We have five syntactic categories: signatures, contexts, kinds, families or types, and objects. This pseudo-syntax is, essentially, that of LF(*cf.* , for instance [15]), with the removal of abstraction in families, and the addition of a *lock constructor* ($\mathcal{L}^{\mathcal{P}}_{N,\sigma}[-]$) on families and objects, and a corresponding *lock destructor* ($\mathcal{U}^{\mathcal{P}}_{N,\sigma}[-]$) on objects Both the lock and unlock constructors are parametrized over a unary logical predicate $\mathcal{P}$, which is defined on derivable type judgements of the form $\Gamma \vdash_{\Sigma} N : \sigma$. The entire $\mathsf{LF}_{\mathcal{P}}$ system is parameterized over a finite set of such predicates and as these predicates are external by nature, they are not formalized explicitly. More comments are provided in Section 4. In [18], we provide pseudo-code for a number of predicates

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

Figure 1: The pseudo-syntax of $\mathsf{LF}_{\mathcal{P}}$

which we have found to be useful, both in general, and for the encodings which we present. However, these predicates are required to satisfy certain well-behavedness conditions, which will be presented in Section 3, in order to ensure subject reduction of the system, while strong normalization and confluence always hold. For the sake of notational completeness, the list of external predicates should also appear in the signature, but we have chosen to omit it in order to increase readability.

Notational conventions and auxiliary definitions. Throughout the paper, we will be using the following notation: $M, N, \ldots \in \mathcal{O}$ denote objects, $f, g, \ldots$ denote object constants, $x, y, z, \ldots$ denote object variables, $\sigma, \tau, \rho, \ldots \in \mathcal{F}$ denote types, $a, b, \ldots$ denote constant types, $K \in \mathcal{K}$ denotes kinds, $\Gamma \in \mathcal{C}$ denotes contexts, $\Sigma \in \mathcal{S}$ denotes signatures, and $\mathcal{P}$ denotes predicates. We refer to $\mathcal{L}$ as the *lock symbol*, and to $\mathcal{U}$ as the *unlock symbol*. We will be using T to denote any term of the calculus (kind, family, or object), where, in some cases, the syntactic category to which T can belong will be clear from the context. We suppose that, in the context $\Gamma, x{:}\sigma$, the variable x does not occur free in Γ or in σ. We will be working modulo α-conversion and Barendregt's variable convention. We define the notions of the domain of a signature, the domain of a context, free and bound variables of a term, as well as substitution in the Appendix. Finally, we will be using $\equiv$ to denote syntactic identity on terms. All of the symbols can appear indexed.

2.1 The $\mathsf{LF}_{\mathcal{P}}$ Type System

The type system for $\mathsf{LF}_{\mathcal{P}}$, presented in Figure 2, proves judgements of the shape:

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

In $\mathsf{LF}_{\mathcal{P}}$, we consider only the terms obtained after a finite number of application of the typing rules of the system. In such terms, each symbol (such as a constant, a variable, a lock, or an unlock) can appear only a finite number of times. Also, we denote by $\Gamma \vdash_{\Sigma} \alpha$ any typing judgement $\Gamma \vdash_{\Sigma} T : T'$ or $\Gamma \vdash_{\Sigma} T$. In the two latter judgements, T will be referred to as the *subject* of that judgement.

2.2 $\beta\mathcal{L}$-reduction and Definitional Equality in $\mathsf{LF}_{\mathcal{P}}$

In $\mathsf{LF}_{\mathcal{P}}$, we have two types of reduction. The first is the standard β-reduction, while the second is a novel form of reduction, which we call $\mathcal{L}$-reduction. $\mathcal{L}$-reduction, essentially, serves as a lock-releasing mechanism, erasing the $\mathcal{U}$-$\mathcal{L}$ pair in a term of the form $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]]$, thus effectively releasing M. Together, these two reductions combine into $\beta\mathcal{L}$-reduction, denoted by $\twoheadrightarrow_{\beta\mathcal{L}}$, and this combined reduction is the one which we take into account when considering the properties of $\mathsf{LF}_{\mathcal{P}}$. The main one-step $\beta\mathcal{L}$-reduction rules are presented in Figure 3. There, we have the new rule $(\mathcal{L}{\cdot}O{\cdot}Main)$, which is the reduction rule illustrating the desired behavior of the lock and unlock combined - the effective release of a lock by an unlock, i.e. the unlock destructor canceling out the lock constructor. This reduction rule, together with the $(O{\cdot}Unlock)$ and $(O{\cdot}Lock)$ typing rules, provides an elegant mechanism for locking and unlocking objects. A similar reduction rule at the level of types is not required, because applying the unlock destructor to a term automatically unlocks its type, as ensured by the $(O{\cdot}Unlock)$ rule.

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

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

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

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

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

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

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

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

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

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

$$\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \quad \to_{\beta\mathcal{L}} \quad M \qquad (\mathcal{L}{\cdot}O{\cdot}Main)$$

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

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

$$\frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\sigma\, N \to_{\beta\mathcal{L}} \sigma'\, N}\ (F{\cdot}A_1{\cdot}\beta\mathcal{L}) \qquad \frac{N \to_{\beta\mathcal{L}} N'}{\sigma\, N \to_{\beta\mathcal{L}} \sigma\, N'}\ (F{\cdot}A_2{\cdot}\beta\mathcal{L})$$

$$\frac{N \to_{\beta\mathcal{L}} N'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N',\sigma}[\rho]}\ (F{\cdot}\mathcal{L}_1{\cdot}\beta\mathcal{L}) \qquad \frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho]}\ (F{\cdot}\mathcal{L}_2{\cdot}\beta\mathcal{L})$$

$$\frac{\rho \to_{\beta\mathcal{L}} \rho'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho']}\ (F{\cdot}\mathcal{L}_3{\cdot}\beta\mathcal{L})$$

Figure 4: $\beta\mathcal{L}$-closure-under-context for families of $\mathsf{LF}_{\mathcal{P}}$

The rules for one-step closure under context for families are presented in Figure 4, while those for kinds and objects are presented in the Appendix (Figure 15 and Figure 16). Furthermore, we denote the reflexive and transitive closure of $\rightarrow_{\beta\mathcal{L}}$ by $\twoheadrightarrow_{\beta\mathcal{L}}$.

We also introduce $\beta\mathcal{L}$-definitional equality in the standard way, as the reflexive, symmetric, and transitive closure of $\beta\mathcal{L}$-reduction on kinds, families, and objects, as illustrated in the Appendix (Figure 17).

3 Properties of $\mathsf{LF}_{\mathcal{P}}$

In this section, we present the main properties of $\mathsf{LF}_{\mathcal{P}}$. Without any additional assumptions on predicates, the type system is strongly normalizing and confluent. The former follows from strong normalization of LF (see [15]), 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 $\mathcal{P}$ in the set satisfies the following conditions:

Closure under signature and context weakening and permutation:

1. If Σ and Ω are valid signatures such that $\Sigma \subseteq \Omega$, and $\mathcal{P}(\Gamma \vdash_\Sigma \alpha)$ holds, then $\mathcal{P}(\Gamma \vdash_\Omega \alpha)$ also holds.
2. If Γ and Δ are valid contexts such that $\Gamma \subseteq \Delta$, and $\mathcal{P}(\Gamma \vdash_\Sigma \alpha)$ holds, then $\mathcal{P}(\Delta \vdash_\Sigma \alpha)$ also holds.

Closure under substitution:

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

Closure under reduction:

1. If $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$ holds, and $N \rightarrow_{\beta\mathcal{L}} N'$ holds, then $\mathcal{P}(\Gamma \vdash_\Sigma N' : \sigma)$ also holds.
2. If $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$ holds, and $\sigma \rightarrow_{\beta\mathcal{L}} \sigma'$ holds, then $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma')$ also holds.

3.1 Strong Normalization

In this section, we prove that $\mathsf{LF}_{\mathcal{P}}$ is strongly normalizing w.r.t. $\beta\mathcal{L}$-reduction. For this, we rely on the strong normalization of LF, as proven in [15]. First, we introduce the function ${}^{-\mathcal{UL}} : \mathsf{LF}_{\mathcal{P}} \rightarrow \mathsf{LF}$, which maps $\mathsf{LF}_{\mathcal{P}}$ terms into LF terms by deleting the $\mathcal{L}$ and $\mathcal{U}$ symbols. The proof then proceeds by contradiction. We assume that there exists a term T with an infinite $\beta\mathcal{L}$-reduction sequence. Next, we prove that only a finite number of β-reductions can be performed within any given $\mathsf{LF}_{\mathcal{P}}$ term T. From this, we deduce that, in order for T to have an infinite $\beta\mathcal{L}$-reduction sequence, it must have an infinite $\mathcal{L}$-sequence, which we show to be impossible, obtaining the desired contradiction. Therefore, we begin with the definition of the function ${}^{-\mathcal{UL}} : \mathsf{LF}_{\mathcal{P}} \rightarrow \mathsf{LF}$:

1. $\mathsf{Type}^{-\mathcal{UL}} = \mathsf{Type};\ \ a^{-\mathcal{UL}} = a;\ \ c^{-\mathcal{UL}} = c;\ \ x^{-\mathcal{UL}} = x;$
2. $(\Pi x{:}\sigma.T)^{-\mathcal{UL}} = \Pi x{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}};$
3. $(\lambda x{:}\sigma.T)^{-\mathcal{UL}} = \lambda x{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}};$
4. $(T\,M)^{-\mathcal{UL}} = T^{-\mathcal{UL}}\,M^{-\mathcal{UL}};$
5. $(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[T])^{-\mathcal{UL}} = (\lambda x_f{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}})\,N^{-\mathcal{UL}};$
6. $(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[T])^{-\mathcal{UL}} = (\lambda x_f{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}})\,N^{-\mathcal{UL}}$

where x_f is a variable which *does not* have free occurrences in T. Its purpose is to preserve the N and σ that appear in the subscript of the $\mathcal{L}$ and $\mathcal{U}$ symbols, while still being able to β-reduce to T in one step, which is in line with the main purpose of ${}^{-\mathcal{UL}}$, *i.e.* the deletion of locks and unlocks from an $\mathsf{LF}_{\mathcal{P}}$ term. We naturally extend ${}^{-\mathcal{UL}}$ to signatures and contexts of $\mathsf{LF}_{\mathcal{P}}$, obtaining signatures and contexts of LF:

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

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

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

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

Proposition 1 (Connecting $\to_{\beta\mathcal{L}}$ in $\mathsf{LF}_{\mathcal{P}}$, $\twoheadrightarrow_\beta$ in LF, and ${}^{-\mathcal{UL}}$)**.**

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

A direct consequence of this is:

Proposition 2 (Connecting $=_{\beta\mathcal{L}}$ in $\mathsf{LF}_{\mathcal{P}}$, $=_\beta$ in LF, and ${}^{-\mathcal{UL}}$)**.**

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

Furthermore, the following holds:

Proposition 3. *The function ${}^{-\mathcal{UL}}$ maps derivable judgements of $\mathsf{LF}_{\mathcal{P}}$ into derivable judgements of LF.*

Proof. All three of these propositions are proven simultaneously, by induction on the structure of the derivation of the reduction, the structure of the derivation of the equivalence, and the structure of the derivation of the judgement. Here, we will present the relevant cases, while the remaining ones are handled similarly.

- For Proposition 1, let $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N',\sigma}[\rho]$ from $N \to_{\beta\mathcal{L}} N'$, using the rule $(F{\cdot}\mathcal{L}_1{\cdot}\beta\mathcal{L})$. From the induction hypothesis (IH), we have that $N^{-\mathcal{UL}} \twoheadrightarrow_\beta N'^{-\mathcal{UL}}$, while the goal we are looking for is $(\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}} \twoheadrightarrow_\beta (\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N'^{-\mathcal{UL}}$, which follows immediately from the IH, and the rules for closure under context for β-reduction in LF.

- For Proposition 1, let $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x]$, using the rule $(\beta{\cdot}O{\cdot}Main)$. Here, we have that the goal is $(\lambda x{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}} \twoheadrightarrow_{\beta\mathcal{L}} M^{-\mathcal{UL}}[N^{-\mathcal{UL}}/x]$, which is, in fact, an instance of the main one-step β-reduction in LF.

- For Proposition 1, let $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} M$, using the rule ($\mathcal{L}$·$O$·$Main$). Here, we have that the goal is $(\lambda x_f{:}\sigma^{-\mathcal{UL}}.(\lambda y_f{:}\sigma^{-\mathcal{UL}}.M)\, N)\, N \twoheadrightarrow_\beta M^{-\mathcal{UL}}$, which we obtain by applying the main one-step β-reduction rule of LF, bearing in mind the nature of the choice of x_f and y_f.

- For Proposition 2, let us have that $K{=}_{\beta\mathcal{L}}K'$, from $K \to_{\beta\mathcal{L}} K'$, using the rule ($\beta\mathcal{L}$·$Eq$·$Main$). From the IH for Proposition 1, we have that $K^{-\mathcal{UL}} \twoheadrightarrow_\beta K'^{-\mathcal{UL}}$ in LF, from which we trivially obtain that $K^{-\mathcal{UL}}{=}_\beta K'^{-\mathcal{UL}}$ in LF.

- For Proposition 3, let us have that $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$, from $\Gamma \vdash_\Sigma \rho : \mathsf{Type}$ and $\Gamma \vdash_\Sigma N : \sigma$, using the rule ($F$·$Lock$). From the IH, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \rho^{-\mathcal{UL}} : \mathsf{Type}$, and $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} N^{-\mathcal{UL}} : \sigma^{-\mathcal{UL}}$, while the goal is $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \mathsf{Type}$. From the Subderivation property of LF, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \sigma^{-\mathcal{UL}} : \mathsf{Type}$, and, given that we can, without loss of generality, assume that $x_f \notin \mathsf{Dom}(\Gamma^{-\mathcal{UL}})$, we obtain, using the Weakening property of LF, that $\Gamma^{-\mathcal{UL}}, x_f : \sigma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \rho^{-\mathcal{UL}} : \mathsf{Type}$, and, from there, that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}} : \Pi x_f{:}\sigma^{-\mathcal{UL}}.\mathsf{Type}$. Now, by using the application formation rule of LF, we obtain that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \mathsf{Type}[N^{-\mathcal{UL}}/x_f]$. However, this is our claim, as, since no substitutions can occur in Type, we have that $\mathsf{Type}[N^{-\mathcal{UL}}/x_f] \equiv \mathsf{Type}$.

- For Proposition 3, let us have that $\Gamma \vdash_\Sigma \sigma : K'$, from $\Gamma \vdash_\Sigma \sigma : K$, $\Gamma \vdash_\Sigma K$, and $K{=}_{\beta\mathcal{L}}K'$, using the rule ($F$·$Conv$). From the IHs, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \sigma^{-\mathcal{UL}} : K^{-\mathcal{UL}}$, $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} K'^{-\mathcal{UL}}$, and $K{=}_\beta K'$, from which we obtain our goal, $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \sigma^{-\mathcal{UL}} : K'^{-\mathcal{UL}}$, by using the LF conversion rule for families.

- For Proposition 3, let us have that $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, from $\Gamma \vdash_\Sigma M : \rho$ and $\Gamma \vdash_\Sigma N : \sigma$, using the rule ($O$·$Lock$). From the IH, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} M^{-\mathcal{UL}} : \rho^{-\mathcal{UL}}$ and $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} N^{-\mathcal{UL}} : \sigma^{-\mathcal{UL}}$, while the goal is $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda x_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : (\lambda y_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}}$. First, as earlier, we can obtain that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \lambda x_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}} : \Pi x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}}$. Now, by using the application formation rule of LF, we obtain that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda x_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \rho^{-\mathcal{UL}}[N^{-\mathcal{UL}}/x_f]$. However, what we actually have is that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda x_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \rho^{-\mathcal{UL}}$, as, by the choice of x_f, we have that $\rho^{-\mathcal{UL}}[N/x_f] \equiv \rho^{-\mathcal{UL}}$. Also, in a similar manner as before, we can obtain that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda y_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \mathsf{Type}$. Now, as, by the choice of y_f, we have that $\rho^{-\mathcal{UL}}{=}_\beta(\lambda y_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}}$, we can use the LF conversion rule for objects to obtain our claim.

- For Proposition 3, let us have that $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho$, from $\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, $\Gamma \vdash_\Sigma N : \sigma$, and $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$, using the rule ($O$·$Unlock$). From the IH, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} M^{-\mathcal{UL}} : (\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}}$ and $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} N^{-\mathcal{UL}} : \sigma^{-\mathcal{UL}}$, while the goal which we would like to prove is $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda y_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \rho^{-\mathcal{UL}}$. Similarly to the previous item, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \rho^{-\mathcal{UL}} : \mathsf{Type}$ and that $(\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}}{=}_\beta\rho^{-\mathcal{UL}}$, and we can use the LF conversion rule for objects to obtain that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} M^{-\mathcal{UL}} : \rho^{-\mathcal{UL}}$. Finally, as we have that $M^{-\mathcal{UL}}{=}_\beta(\lambda y_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}}$, we obtain the desired claim by using the Subject Reduction property of LF.

□

As a consequence of Proposition 3, we have that the function $^{-\mathcal{UL}}$ maps well-typed terms of $\mathsf{LF}_\mathcal{P}$ into well-typed terms of LF. Next, we will denote the maximum number of β-reductions which can be executed in a given (either LF- or $\mathsf{LF}_\mathcal{P}$-) term T as $\max_\beta(T)$. Now, we can notice that $\mathcal{L}$-reductions cannot create entirely new β-redexes, but can only "unlock" potential β-redexes, *i.e.* expressions of the form $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{P}_{N,\sigma}[\lambda x{:}\tau.M]]\, T$, arriving at $\lambda x{:}\tau.M\, T$, which is a β-redex. Also, this resulting β-redex will be present in $(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{P}_{N,\sigma}[\lambda x{:}\tau.M]]\, T)^{-\mathcal{UL}}$. Therefore, we have that, for any $\mathsf{LF}_\mathcal{P}$-term T, it holds that $\max_\beta(T) \leq \max_\beta(T^{-\mathcal{UL}})$. As LF is strongly normalizing, we have that $\max_\beta(T^{-\mathcal{UL}})$ is finite, therefore forcing $\max_\beta(T)$ into being finite, leading to the following proposition:

Proposition 4. *Only finitely many β-reductions can occur within the maximal reduction sequence of any $\mathsf{LF}_\mathcal{P}$-term. There is no $\mathsf{LF}_\mathcal{P}$-term T with an infinite number of β-reductions in its maximal reduction sequence.*

Next, we notice that any $\mathsf{LF}_{\mathcal{P}}$-term has only finitely many $\mathcal{L}$-redexes before any reductions take place, and that this number can only be increased through β-reductions, and only by a finite amount per β-reduction. However, if we were to have an $\mathsf{LF}_{\mathcal{P}}$-term T which has an infinite reduction sequence, then within this sequence, there would need to be infinitely many $\mathcal{L}$-reductions, since, due to Proposition 4, the number of β-reductions in this sequence has to be finite. On the other hand, with the number of β-reductions in the sequence being finite, it would not be possible to reach infinitely many $\mathcal{L}$-reductions, and such a term T cannot exist in $\mathsf{LF}_{\mathcal{P}}$. Therefore, we have the Strong Normalization theorem:

Theorem 1 (Strong normalization of $\mathsf{LF}_{\mathcal{P}}$)**.**

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

3.2 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* $N \to_{\beta\mathcal{L}} N'$*, then* $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M[N'/x]$*.*
2. *If* $M \to_{\beta\mathcal{L}} M'$*, then* $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$*.*

Next, we proceed to prove local confluence:

Lemma 2 (Local confluence of $\mathsf{LF}_{\mathcal{P}}$)**.** *$\beta\mathcal{L}$-reduction is locally confluent, i.e.:*

1. *If* $K \to_{\beta\mathcal{L}} K'$ *and* $K \to_{\beta\mathcal{L}} K''$*, then there exists a* K''' *such that* $K' \twoheadrightarrow_{\beta\mathcal{L}} K'''$ *and* $K'' \twoheadrightarrow_{\beta\mathcal{L}} K''$*.*
2. *If* $\sigma \to_{\beta\mathcal{L}} \sigma'$ *and* $\sigma \to_{\beta\mathcal{L}} \sigma''$*, then there exists a* σ''' *such that* $\sigma' \twoheadrightarrow_{\beta\mathcal{L}} \sigma'''$ *and* $\sigma'' \twoheadrightarrow_{\beta\mathcal{L}} \sigma'''$*.*
3. *If* $M \to_{\beta\mathcal{L}} M'$ *and* $M \to_{\beta\mathcal{L}} M''$*, then there exists an* M''' *such that* $M' \twoheadrightarrow_{\beta\mathcal{L}} M'''$ *and* $M'' \twoheadrightarrow_{\beta\mathcal{L}} M'''$*.*

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

1. Let us have, by the base reduction rule $(\beta{\cdot}Main)$, that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x]$. Let us also have that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} (\lambda x{:}\sigma'.M)\, N$, from $\sigma \to_{\beta\mathcal{L}} \sigma'$, by the reduction rules $(O{\cdot}\lambda_1{\cdot}\beta\mathcal{L})$ and $(O{\cdot}App_1{\cdot}\beta\mathcal{L})$. In this case, we will show that the required conditions are met for $M''' \equiv M[N/x]$. Indeed, by the definition of $\twoheadrightarrow_{\beta\mathcal{L}}$, we have that $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M[N/x]$, and also, by the reduction rule $(\beta{\cdot}Main)$, we have that $(\lambda x{:}\sigma'.M)\, N \to_{\beta\mathcal{L}} M[N/x]$, effectively having $(\lambda x{:}\sigma'.M)\, N \twoheadrightarrow_{\beta\mathcal{L}} M[N/x]$.
2. Let us have, by the base reduction rule $(\beta{\cdot}Main)$, that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x]$. Let us also have that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} (\lambda x{:}\sigma.M')\, N$, from $M \to_{\beta\mathcal{L}} M'$, by the reduction rules $(O{\cdot}\lambda_2{\cdot}\beta\mathcal{L})$ and $(O{\cdot}App_1{\cdot}\beta\mathcal{L})$. In this case, we will show that the required conditions are met for $M''' \equiv M'[N/x]$. By the reduction rule $(\beta{\cdot}Main)$, we have that $(\lambda x{:}\sigma.M')\, N \to_{\beta\mathcal{L}} M'[N/x]$, from which we obtain $(\lambda x{:}\sigma.M')\, N \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$, while we obtain that $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$ from part 2 of Lemma 1.
3. Let us have, by the base reduction rule $(\beta{\cdot}Main)$, that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x]$. Let us also have that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} (\lambda x{:}\sigma.M)\, N'$, from $N \to_{\beta\mathcal{L}} N'$, by the reduction rule $(O{\cdot}App_2{\cdot}\beta\mathcal{L})$. In this case, we will show that the required conditions are met for $M''' \equiv M[N'/x]$. By the reduction rule $(\beta{\cdot}Main)$, we have that $(\lambda x{:}\sigma.M)\, N' \to_{\beta\mathcal{L}} M[N'/x]$, from which we obtain that $(\lambda x{:}\sigma.M)\, N' \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$, while we obtain that $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M[N'/x]$ from part 1 of Lemma 1.

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

□

Having proven local confluence, finally, from Theorem 1, Lemma 2 and Newman's Lemma, we obtain the confluence theorem for $\mathsf{LF}_{\mathcal{P}}$:

Theorem 2 (Confluence of $\mathsf{LF}_{\mathcal{P}}$)**.** *$\beta\mathcal{L}$-reduction is confluent, i.e.:*

1. *If $K \twoheadrightarrow_{\beta\mathcal{L}} K'$ and $K \twoheadrightarrow_{\beta\mathcal{L}} K''$, then there exists a K''' such that $K' \twoheadrightarrow_{\beta\mathcal{L}} K'''$ and $K'' \twoheadrightarrow_{\beta\mathcal{L}} K'''$.*
2. *If $\sigma \twoheadrightarrow_{\beta\mathcal{L}} \sigma'$ and $\sigma \twoheadrightarrow_{\beta\mathcal{L}} \sigma''$, then there exists a σ''' such that $\sigma' \twoheadrightarrow_{\beta\mathcal{L}} \sigma'''$ and $\sigma'' \twoheadrightarrow_{\beta\mathcal{L}} \sigma'''$.*
3. *If $M \twoheadrightarrow_{\beta\mathcal{L}} M'$ and $M \twoheadrightarrow_{\beta\mathcal{L}} M''$, then there exists an M''' such that $M' \twoheadrightarrow_{\beta\mathcal{L}} M'''$ and $M'' \twoheadrightarrow_{\beta\mathcal{L}} M'''$.*

3.3 Subject Reduction

We begin by proving several auxiliary lemmas and propositions:

Lemma 3 (Inversion properties)**.**

1. *If $\Pi x{:}\sigma.T{=}_{\beta\mathcal{L}}T''$, then $T'' \equiv \Pi x{:}\sigma.'T'$, for some σ', T', such that $\sigma'{=}_{\beta\mathcal{L}}\sigma$, and $T'{=}_{\beta\mathcal{L}}T$.*
2. *If $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]{=}_{\beta\mathcal{L}}\theta$, then $\theta \equiv \mathcal{L}^{\mathcal{P}}_{N',\sigma'}[\rho']$, for some N', σ', and ρ', such that $N'{=}_{\beta\mathcal{L}}N$, $\sigma'{=}_{\beta\mathcal{L}}\sigma$, and $\rho'{=}_{\beta\mathcal{L}}\rho$.*
3. *If $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, then $\Gamma \vdash_\Sigma M : \rho$.*
4. *If $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau$, then $\Gamma, x{:}\sigma \vdash_\Sigma M : \tau$.*

Proof. The first two items are proven directly, by inspection of the rules for $\beta\mathcal{L}$-closure-under-context for kinds and types, while the third item is proven by using the rule ($F{\cdot}Lock$), and the second item. The fourth property follows directly from the typing and conversion rules, as well as the first item. □

By induction on the structure of the derivation, independently of the previous Proposition, we have:

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

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

From this point on, we will assume that the first requirement for the well-behavedness of predicates, namely the *closure under signature and context weakening and permutation*, holds. With this in place, we prove the following propositions by induction on the structure of the derivation:

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

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

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

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

From this point on, we will assume that the second requirement for the well-behavedness of predicates, the *closure under cut*, holds as well. We prove the following propositions by induction on the structure of the derivation:

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

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

1. *If* $\Gamma \vdash_\Sigma \sigma : K_1$ *and* $\Gamma \vdash_\Sigma \sigma : K_2$*, then* $\Gamma \vdash_\Sigma K_1 =_{\beta\mathcal{L}} K_2$*.*
2. *If* $\Gamma \vdash_\Sigma M : \sigma_1$ *and* $\Gamma \vdash_\Sigma M : \sigma_2$*, then* $\Gamma \vdash_\Sigma \sigma_1 =_{\beta\mathcal{L}} \sigma_2$*.*

Finally, for Subject Reduction, we require that the third requirements for the well-behavedness of predicates, namely the *closure under definitional equality*, also holds:

Theorem 3 (Subject reduction of $\mathsf{LF}_\mathcal{P}$)**.** *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* $\Gamma \vdash_\Sigma M' : \sigma$*.*

Proof. Here, we prove Subject Reduction of a slightly extended type system. We consider the type system in which the rules ($F{\cdot}Lock$), ($O{\cdot}Lock$), and ($O{\cdot}Unlock$) all have an additional premise $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$, while the rule ($O{\cdot}Unlock$) also has another additional premise $\Gamma \vdash_\Sigma \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho] : \mathsf{Type}$, as shown in Figure 5.

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

1. We have that $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M\, N : \tau[N/x]$, by the rule ($O{\cdot}App$), from $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau$, and $\Gamma \vdash_\Sigma N : \sigma$, and that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x]$ by the rule ($\beta{\cdot}Main$). From Proposition 3, we get that $\Gamma, x{:}\sigma \vdash_\Sigma M : \tau$, and from this and $\Gamma \vdash_\Sigma N : \sigma$, we obtain the required $\Gamma \vdash_\Sigma M[N/x] : \tau[N/x]$, by an application of Proposition 8.
2. We have that $\Gamma \vdash_\Sigma \mathcal{U}^\mathcal{P}_{N,\sigma}[\mathcal{L}^\mathcal{P}_{N,\sigma}[M]] : \rho$, by the rule ($O{\cdot}Unlock$), from $\Gamma \vdash_\Sigma \mathcal{L}^\mathcal{P}_{N,\sigma}[M] : \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho]$, $\Gamma \vdash_\Sigma N : \sigma$, $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$, and $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$, and that $\mathcal{U}^\mathcal{P}_{N,\sigma}[\mathcal{L}^\mathcal{P}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} M$ by the rule ($\mathcal{L}{\cdot}Main$). Here, we obtain the required $\Gamma \vdash_\Sigma M : \rho$ directly, using the last two items of Lemma 3.

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

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

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

Figure 5: An extension of $\mathsf{LF}_{\mathcal{P}}$ typing rules for Subject Reduction

3. We have that $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho$, by the rule $(O{\cdot}Unlock)$, from $\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$, $\Gamma \vdash_\Sigma N : \sigma$, $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$, and $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$, and that $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N,\sigma'}[M]$, by the reduction rules for closure under context, from $\sigma \to_{\beta\mathcal{L}} \sigma'$. First, from the induction hypothesis we have that $\Gamma \vdash_\Sigma \sigma' : \mathsf{Type}$, and we also have, from $\sigma \to_\beta : \sigma'$, that $\sigma =_{\beta\mathcal{L}} \sigma'$. From this, using $\Gamma \vdash_\Sigma N : \sigma$, and the rule $(O{\cdot}Conv)$, we obtain that $\Gamma \vdash_\Sigma N : \sigma'$. Next, since $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$ could only have been obtained by the type system rule $(F{\cdot}Lock)$, from $\Gamma \vdash_\Sigma \rho : \mathsf{Type}$ and $\Gamma \vdash_\Sigma N : \sigma$, and since we have $\Gamma \vdash_\Sigma N : \sigma'$, we obtain that $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho] : \mathsf{Type}$. From this, given $\sigma =_{\beta\mathcal{L}} \sigma'$, we obtain that $\mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho]$, and since we already have that $\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, we can use the type system rule $(O{\cdot}Conv)$ to obtain $\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho]$. Finally, by the well-behavedness requirements for the predicates, we have that $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma')$ holds, and we can now use the type system rule $(O{\cdot}Unlock)$ to obtain the required $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma'}[M] : \rho$. Here, we can notice that there are steps in this proof (in which we obtain $\Gamma \vdash_\Sigma \sigma' : \mathsf{Type}$, and $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$), which could not have been made had the original system not been extended for this theorem.

Now, we can prove straightforwardly that $\Gamma \vdash_\Sigma \alpha$ in the extended system *iff* $\Gamma \vdash_\Sigma \alpha$ in the original $\mathsf{LF}_{\mathcal{P}}$ 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 $\mathsf{LF}_{\mathcal{P}}$ system. □

4 The Expressive Power of $\mathsf{LF}_{\mathcal{P}}$

Various natural questions arise as to the expressive power of $\mathsf{LF}_{\mathcal{P}}$, and we will here outline the answers to some of them:

- $\mathsf{LF}_{\mathcal{P}}$ is decidable, if the predicates are decidable; this can be proven as usual.
- If a predicate is *definable in* LF, *i.e.* if 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 recursively enumerable 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 deep. As an illustration, we can consider, for example, the well-behaved predicate "M, N are two different closed normal forms", which can be immediately expressed in $\mathsf{LF}_{\mathcal{P}}$.
- One may ask what relation is there between the LF encodings of, say, Modal Logics, discussed in [2, 12], and the encodings which appear in this paper (see Section 6.3 below). The former

$$
\begin{array}{rcl@{\qquad}rcl@{\qquad}l}
\alpha & \in & \mathcal{A} & \alpha & ::= & a \mid \alpha\, N & \textit{Atomic Families} \\
\sigma, \tau, \rho & \in & \mathcal{F} & \sigma & ::= & \alpha \mid \Pi x{:}\sigma.\tau \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] & \textit{Canonical Families} \\
A & \in & \mathcal{A}_o & A & ::= & c \mid x \mid A\, M \mid \mathcal{U}^{\mathcal{P}}_{N,\sigma}[A] & \textit{Atomic Objects} \\
M, N & \in & \mathcal{O} & M & ::= & A \mid \lambda x{:}\sigma.M \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] & \textit{Canonical Objects}
\end{array}
$$

Figure 6: $\mathsf{CLF}_{\mathcal{P}}$ Syntax

essentially correspond to the internal encoding of the predicates that are utilized in Section 6.3. 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.

- 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 the 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 such a separation explicit.

- The main advantage of having externally verified predicates amounts to a smoother encoding (the signature is not cluttered by auxiliary notions and mechanisms needed to implement the predicate). This allows performance optimization, if the external system used to verify the predicate is an optimized tool specifically designed for the issue at hand (*e.g.* analytic tableaux methods for propositional formulæ).

5 The Canonical $\mathsf{LF}_{\mathcal{P}}$ Framework

In this section, we present a *canonical* version of $\mathsf{LF}_{\mathcal{P}}$, *i.e.* $\mathsf{CLF}_{\mathcal{P}}$, in the style of [36, 16]. 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 $\mathsf{CLF}_{\mathcal{P}}$ 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 $\mathsf{LF}_{\mathcal{P}}$. $\mathsf{CLF}_{\mathcal{P}}$ will be the basis for proving the adequacy of the encodings which are presented in Section 6. 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 $\mathsf{CLF}_{\mathcal{P}}$ is not closed under full η-expansion, it is still powerful enough for one to be able to obtain all relevant adequacy results.

5.1 Syntax and Type System for $\mathsf{CLF}_{\mathcal{P}}$

The pseudo-syntax of $\mathsf{CLF}_{\mathcal{P}}$ is presented in Figure 6, while the type system for $\mathsf{CLF}_{\mathcal{P}}$ is shown in Figure 7, with the note that the formation rules and judgements related to signatures, contexts, and kinds are the same as in $\mathsf{LF}_{\mathcal{P}}$, and have been omitted. The type system for $\mathsf{CLF}_{\mathcal{P}}$, together with the first three judgements of $\mathsf{LF}_{\mathcal{P}}$ (see Section 2.1), proves judgements of the shape:

$$
\begin{array}{lll@{\qquad}l}
\Gamma & \vdash_{\Sigma} & \sigma\ \mathsf{Type} & \sigma \text{ is a canonical family in } \Gamma \text{ and } \Sigma \\
\Gamma & \vdash_{\Sigma} & \alpha \Rightarrow K & K \text{ is the kind of the atomic family } \alpha \text{ in } \Gamma \text{ and } \Sigma \\
\Gamma & \vdash_{\Sigma} & M \Leftarrow \sigma & M \text{ is a canonical term of type } \sigma \text{ in } \Gamma \text{ and } \Sigma \\
\Gamma & \vdash_{\Sigma} & A \Rightarrow \sigma & \sigma \text{ is the type of the atomic term } A \text{ in } \Gamma \text{ and } \Sigma
\end{array}
$$

The predicates $\mathcal{P}$ in $\mathsf{CLF}_{\mathcal{P}}$ are defined on judgements $\Gamma \vdash_{\Sigma} M \Leftarrow \sigma$. The type system makes use, in the rules ($A{\cdot}App$) and ($F{\cdot}App$), of 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). The simple type ρ of M is obtained via the erasure function (presented in Fig. 8), which maps dependent into simple types. The rules for the hereditary substitution

Atomic Family rules

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

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

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

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

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

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

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

Canonical Object rules

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

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

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

Figure 7: The $\mathsf{CLF}_{\mathcal{P}}$ Type System

$$\frac{}{(a)^- = a} \qquad \frac{(\alpha)^- = \rho}{(\alpha M)^- = \rho} \qquad \frac{(\sigma_2)^- = \rho_2 \quad (\sigma)^- = \rho}{(\Pi x{:}\sigma_2.\sigma)^- = \rho_2 \to \rho} \qquad \frac{(\sigma')^- = \rho'}{(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\sigma'])^- = \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho']}$$

Figure 8: Erasure to simple types

judgement appear in Figure 9 and Figure 10. At this point, we remind the reader that we are still abiding by Barendregt's hygiene condition.

Notice that, in the rule $(O{\cdot}Atom)$ of the type system (see Fig. 7), 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. In Definition 3 below, we formalize the notion of η-expansion of a judgement, together with correspondence theorems between $\mathsf{LF}_{\mathcal{P}}$ and $\mathsf{CLF}_{\mathcal{P}}$.

5.2 Properties of $\mathsf{CLF}_{\mathcal{P}}$

We start by studying basic properties of hereditary substitution and the type system.

Lemma 4 (Decidability of hereditary substitution)**.**

1. *For any T in $\{\mathcal{K}, \mathcal{A}, \mathcal{F}, \mathcal{O}, \mathcal{C}\}$, 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'.*

Substitution in Kinds

$$\frac{}{\mathsf{Type}[M_0/x_0]^K_{\rho_0} = \mathsf{Type}}\ (\mathcal{S}{\cdot}K{\cdot}Type) \quad \frac{\sigma[M_0/x_0]^F_{\rho_0} = \sigma' \quad K[M_0/x_0]^K_{\rho_0} = K'}{(\Pi x{:}\sigma.K)[M_0/x_0]^K_{\rho_0} = \Pi x{:}\sigma.'K'}\ (\mathcal{S}{\cdot}K{\cdot}Pi)$$

Substitution in Atomic Families

$$\frac{}{a[M_0/x_0]^f_{\rho_0} = a}\ (\mathcal{S}{\cdot}F{\cdot}Const) \quad \frac{\alpha[M_0/x_0]^f_{\rho_0} = \alpha' \quad M[M_0/x_0]^O_{\rho_0} = M'}{(\alpha M)[M_0/x_0]^f_{\rho_0} = \alpha' M'}\ (\mathcal{S}{\cdot}F{\cdot}App)$$

Substitution in Canonical Families

$$\frac{\alpha[M_0/x_0]^f_{\rho_0} = \alpha'}{\alpha[M_0/x_0]^F_{\rho_0} = \alpha'}\ (\mathcal{S}{\cdot}F{\cdot}Atom) \quad \frac{\sigma_1[M_0/x_0]^F_{\rho_0} = \sigma_1' \quad \sigma_2[M_0/x_0]^F_{\rho_0} = \sigma_2'}{(\Pi x{:}\sigma_1.\sigma_2)[M_0/x_0]^F_{\rho_0} = \Pi x{:}\sigma_1'.\sigma_2'}\ (\mathcal{S}{\cdot}F{\cdot}Pi)$$

$$\frac{\sigma_1[M_0/x_0]^F_{\rho_0} = \sigma_1' \quad M_1[M_0/x_0]^O_{\rho_0} = M_1' \quad \sigma_2[M_0/x_0]^F_{\rho_0} = \sigma_2'}{\mathcal{L}^{\mathcal{P}}_{M_1,\sigma_1}[\sigma_2][M_0/x_0]^F_{\rho_0} = \mathcal{L}^{\mathcal{P}}_{M_1',\sigma_1'}[\sigma_2']}\ (\mathcal{S}{\cdot}F{\cdot}Lock)$$

Figure 9: Hereditary substitution, kinds and families

Substitution in Atomic Objects

$$\frac{}{c[M_0/x_0]^o_{\rho_0} = c}\ (\mathcal{S}{\cdot}O{\cdot}Const)$$

$$\frac{}{x_0[M_0/x_0]^o_{\rho_0} = M_0 : \rho_0}\ (\mathcal{S}{\cdot}O{\cdot}Var{\cdot}H) \quad \frac{x \neq x_0}{x[M_0/x_0]^o_{\rho_0} = x}\ (\mathcal{S}{\cdot}O{\cdot}Var)$$

$$\frac{A_1[M_0/x_0]^o_{\rho_0} = \lambda x{:}\rho_2.M_1' : \rho_2 \to \rho \quad M_2[M_0/x_0]^O_{\rho_0} = M_2' \quad M_1'[M_2'/x]^O_{\rho_2} = M'}{(A_1 M_2)[M_0/x_0]^o_{\rho_0} = M' : \rho}\ (\mathcal{S}{\cdot}O{\cdot}App{\cdot}H)$$

$$\frac{A_1[M_0/x_0]^o_{\rho_0} = A_1' \quad M_2[M_0/x_0]^O_{\rho_0} = M_2'}{(A_1 M_2)[M_0/x_0]^o_{\rho_0} = A_1' M_2'}\ (\mathcal{S}{\cdot}O{\cdot}App)$$

$$\frac{\sigma[M_0/x_0]^F_{\rho_0} = \sigma' \quad M[M_0/x_0]^O_{\rho_0} = M' \quad A[M_0/x_0]^o_{\rho_0} = \mathcal{L}^{\mathcal{P}}_{M',\sigma'}[M_1] : \mathcal{L}^{\mathcal{P}}_{M',\sigma'}[\rho]}{\mathcal{U}^{\mathcal{P}}_{M,\sigma}[A][M_0/x_0]^o_{\rho_0} = M_1 : \rho}\ (\mathcal{S}{\cdot}O{\cdot}Unlock{\cdot}H)$$

$$\frac{\sigma[M_0/x_0]^F_{\rho_0} = \sigma' \quad M[M_0/x_0]^O_{\rho_0} = M' \quad A[M_0/x_0]^O_{\rho_0} = A'}{\mathcal{U}^{\mathcal{P}}_{M,\sigma}[A][M_0/x_0]^o_{\rho_0} = \mathcal{U}^{\mathcal{P}}_{M',\sigma'}[A']}\ (\mathcal{S}{\cdot}O{\cdot}Unlock)$$

Substitution in Canonical Objects

$$\frac{A[M_0/x_0]^o_{\rho_0} = A'}{A[M_0/x_0]^O_{\rho_0} = A'}\ (\mathcal{S}{\cdot}O{\cdot}R) \quad \frac{A[M_0/x_0]^o_{\rho_0} = M' : \rho}{A[M_0/x_0]^O_{\rho_0} = M'}\ (\mathcal{S}{\cdot}O{\cdot}R{\cdot}H)$$

$$\frac{M[M_0/x_0]^O_{\rho_0} = M'}{\lambda x{:}\sigma.M[M_0/x_0]^O_{\rho_0} = \lambda x{:}\sigma.M'}\ (\mathcal{S}{\cdot}O{\cdot}Abs)$$

$$\frac{\sigma_1[M_0/x_0]^F_{\rho_0} = \sigma_1' \quad M_1[M_0/x_0]^O_{\rho_0} = M_1' \quad M_2[M_0/x_0]^O_{\rho_0} = M_2'}{\mathcal{L}^{\mathcal{P}}_{M_1,\sigma_1}[M_2][M_0/x_0]^O_{\rho_0} = \mathcal{L}^{\mathcal{P}}_{M_1',\sigma_1'}[M_2']}\ (\mathcal{S}{\cdot}O{\cdot}Lock)$$

Substitution in Contexts

$$\frac{}{\cdot[M_0/x_0]^C_{\rho_0} = \cdot}\ (\mathcal{S}{\cdot}Ctxt{\cdot}Empty)$$

$$\frac{x_0 \neq x \quad x \notin \mathsf{Fv}(M_0) \quad \Gamma[M_0/x_0]^C_{\rho_0} = \Gamma' \quad \sigma[M_0/x_0]^F_{\rho_0} = \sigma'}{\Gamma, x{:}\sigma[M_0/x_0]^C_{\rho_0} = \Gamma', x{:}\sigma'}\ (\mathcal{S}{\cdot}Ctxt{\cdot}Term)$$

Figure 10: Hereditary substitution, objects and contexts

2. *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'.*

Lemma 5 (Head substitution size)**.** *If $A[M_0/x_0]^o_{\rho_0} = M{:}\rho$, then ρ is a subexpression of ρ_0.*

Proof. This Lemma is proven directly, by induction on the derivation of $A[M_0/x_0]_{\rho_0} = M : \rho$. □

Lemma 6 (Uniqueness of substitution and synthesis)**.**

1. *It is not possible that $A[M_0/x_0]^o_{\rho_0} = A'$ and $A[M_0/x_0]^o_{\rho_0} = M{:}\rho$.*
2. *For any T, if $T[M_0/x_0]^m_{\rho_0} = T'$, and $T[M_0/x_0]^m_{\rho_0} = T''$, then $T' = T''$.*
3. *If $\Gamma \vdash_\Sigma \alpha \Rightarrow K$, and $\Gamma \vdash_\Sigma \alpha \Rightarrow K'$, then $K = K'$.*
4. *If $\Gamma \vdash_\Sigma A \Rightarrow \sigma$, and $\Gamma \vdash_\Sigma A \Rightarrow \sigma'$, then $\sigma = \sigma'$.*

Proof. The proof of this Lemma is trivial. It follows directly from the definition of hereditary substitution and the $\mathsf{CLF}_\mathcal{P}$ type system. □

Lemma 7 (Composition of hereditary substitution)**.** *Let us assume that $x \neq x_0, M_0$. Then:*

1. *If $M_2[M_0/x_0]^O_{\rho_0} = M_2'$, $T_1[M_2/x]^m_{\rho_2} = T_1'$, and $T_1[M_0/x_0]^m_{\rho_0} = T_1''$, then there exists a T, such that $T_1'[M_0/x_0]^m_{\rho_0} = T$, and $T_1''[M_2/x]^m_{\rho_2} = M' : \rho$.*
2. *If $M_2[M_0/x_0]^O_{\rho_0} = M_2'$, $A_1[M_2/x]^o_{\rho_2} = M : \rho$, and $A_1[M_0/x_0]^o_{\rho_0} = A$, then there exists an M', such that $M[M_0/x_0]^O_{\rho_0} = M'$, and $A[M_2/x]^o_{\rho_2} = M' : \rho$.*
3. *If $M_2[M_0/x_0]^O_{\rho_0} = M_2'$, $A_1[M_2/x]^o_{\rho_2} = A$, and $A_1[M_0/x_0]^o_{\rho_0} = M : \rho$, then there exists an M', such that $A[M_0/x_0]^o_{\rho_0} = M' : \rho$, and $M[M_2/x]^O_{\rho_2} = M'$.*

By induction on derivations, similar to one in [16] p.14–15, we prove:

Theorem 4 (Transitivity)**.** *Let Σ sig, $\vdash_\Sigma \Gamma_L, x_0{:}\rho_0, \Gamma_R$ and $\Gamma_L \vdash_\Sigma M_0 \Leftarrow \rho_0$, and assume that all predicates are well-behaved. Then*

1. *There exists Γ_R' such that $[M_0/x_0]^C_{\rho_0} = \Gamma_R'$ and $\vdash_\Sigma \Gamma_L, \Gamma_R'$.*
2. *If $\Gamma_L, x_0{:}\rho_0, \Gamma_R \vdash_\Sigma K$ then there exists K' such that $[M_0/x_0]^K_{\rho_0} K = K'$ and $\Gamma_L, \Gamma_R' \vdash_\Sigma K'$.*
3. *If $\Gamma_L, x_0{:}\rho_0, \Gamma_R \vdash_\Sigma \sigma$ Type, then there exists σ' such that $[M_0/x_0]^F_{\rho_0}\sigma = \sigma'$ and $\Gamma_L, \Gamma_R' \vdash_\Sigma \sigma'$ Type.*
4. *If $\Gamma_L, x_0{:}\rho_0, \Gamma_R \vdash_\Sigma \sigma$ Type and $\Gamma_L, x_0{:}\rho_0, \Gamma_R \vdash_\Sigma M \Leftarrow \sigma$, then there exist σ' and M' such that $[M_0/x_0]^F_{\rho_0}\sigma = \sigma'$ and $[M_0/x_0]^O_{\rho_0} M = M'$ and $\Gamma_L, \Gamma_R' \vdash_\Sigma M' \Leftarrow \sigma'$.*

Theorem 5 (Decidability of typing)**.** *If predicates in $\mathsf{CLF}_\mathcal{P}$ are decidable, then all of the judgements of the system are decidable.*

Proof. By induction on the complexity of judgements. □ □

Now we are in the position of making precise the relationships between $\mathsf{CLF}_\mathcal{P}$ and the original $\mathsf{LF}_\mathcal{P}$ system.

Theorem 6 (Soundness)**.** *For any predicate $\mathcal{P}$ of $\mathsf{CLF}_\mathcal{P}$, 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 $\mathsf{CLF}_\mathcal{P}$. Then, we have:*

1. *If Σ sig is derivable in $\mathsf{CLF}_\mathcal{P}$, then Σ sig is derivable in $\mathsf{LF}_\mathcal{P}$.*
2. *If $\vdash_\Sigma \Gamma$ is derivable in $\mathsf{CLF}_\mathcal{P}$, then $\vdash_\Sigma \Gamma$ is derivable in $\mathsf{LF}_\mathcal{P}$.*
3. *If $\Gamma \vdash_\Sigma K$ is derivable in $\mathsf{CLF}_\mathcal{P}$, then $\Gamma \vdash_\Sigma K$ is derivable in $\mathsf{LF}_\mathcal{P}$.*
4. *If $\Gamma \vdash_\Sigma \alpha \Rightarrow K$ is derivable in $\mathsf{CLF}_\mathcal{P}$, then $\Gamma \vdash_\Sigma \alpha : K$ is derivable in $\mathsf{LF}_\mathcal{P}$.*

5. *If* $\Gamma \vdash_\Sigma \sigma$ Type *is derivable in* $\mathsf{CLF}_\mathcal{P}$*, 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* $\mathsf{CLF}_\mathcal{P}$*, then* $\Gamma \vdash_\Sigma A : \sigma$ *is derivable in* $\mathsf{LF}_\mathcal{P}$*.*

7. *If* $\Gamma \vdash_\Sigma M \Leftarrow \sigma$ *is derivable in* $\mathsf{CLF}_\mathcal{P}$*, then* $\Gamma \vdash_\Sigma M : \sigma$ *is derivable in* $\mathsf{LF}_\mathcal{P}$*.*

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

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

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 $\Pi\vec{x}_1{:}\vec{\sigma}_1.\vec{\mathcal{L}}_1[\dots\Pi\vec{x}_n{:}\vec{\sigma}_n.\vec{\mathcal{L}}_n[\alpha]\dots]$, where $\vec{\mathcal{L}}_1, \dots, \vec{\mathcal{L}}_n$ are vectors of locks, if ξ appears in contexts of the form $\vec{\mathcal{U}}_n[(\dots(\vec{\mathcal{U}}_1[\xi\vec{M}_1])\dots)\vec{M}_n]$, where $\vec{M}_1, \dots, \vec{M}_n$, $\vec{\mathcal{U}}_1, \dots, \vec{\mathcal{U}}_n$ have the same arities of the corresponding vectors of Π's and locks.

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

1. 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.

2. A judgement is in *η-lnf* if all terms appearing in it are in η-lnf.

Definition 4 (Well-behaved $\mathsf{CLF}_\mathcal{P}$-predicates)**.** A finite set of predicates $\{\mathcal{P}_i\}_{i\in I}$ is *well-behaved* in $\mathsf{CLF}_\mathcal{P}$ if each $\mathcal{P}$ is *closed under signature*, *context weakening*, *permutation*, and *hereditary substitution*.

Theorem 7 (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* $\mathsf{CLF}_\mathcal{P}$ *with:* $\mathcal{P}(\Gamma \vdash_\Sigma M \Leftarrow \sigma)$ *holds if* $\Gamma \vdash_\Sigma M \Leftarrow \sigma$ *is derivable in* $\mathsf{CLF}_\mathcal{P}$ *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* $\mathsf{LF}_\mathcal{P}$*-derivable, then* Σ sig *is* $\mathsf{CLF}_\mathcal{P}$*-derivable.*

2. *If* $\vdash_\Sigma \Gamma$ *is in η-lnf and is* $\mathsf{LF}_\mathcal{P}$*-derivable, then* $\vdash_\Sigma \Gamma$ *is* $\mathsf{CLF}_\mathcal{P}$*-derivable.*

3. *If* $\Gamma \vdash_\Sigma K$ *is in η-lnf, and is* $\mathsf{LF}_\mathcal{P}$*-derivable, then* $\Gamma \vdash_\Sigma K$ *is* $\mathsf{CLF}_\mathcal{P}$*-derivable.*

4. *If* $\Gamma \vdash_\Sigma \alpha : K$ *is in η-lnf and is* $\mathsf{LF}_\mathcal{P}$*-derivable, then* $\Gamma \vdash_\Sigma \alpha \Rightarrow K$ *is* $\mathsf{CLF}_\mathcal{P}$*-derivable.*

5. *If* $\Gamma \vdash_\Sigma \sigma{:}\mathsf{Type}$ *is in η-lnf and is* $\mathsf{LF}_\mathcal{P}$*-derivable, then* $\Gamma \vdash_\Sigma \sigma$ Type *is* $\mathsf{CLF}_\mathcal{P}$*-derivable.*

6. *If* $\Gamma \vdash_\Sigma A : \alpha$ *is in η-lnf and is* $\mathsf{LF}_\mathcal{P}$*-derivable, then* $\Gamma \vdash_\Sigma A \Rightarrow \alpha$ *is* $\mathsf{CLF}_\mathcal{P}$*-derivable.*

7. *If* $\Gamma \vdash_\Sigma M : \sigma$ *is in η-lnf and is* $\mathsf{LF}_\mathcal{P}$*-derivable, then* $\Gamma \vdash_\Sigma M \Leftarrow \sigma$ *is* $\mathsf{CLF}_\mathcal{P}$*-derivable.*

Proof. We prove all items by mutual induction on the complexity of the judgement, where the complexity of a judgement is given by the sum of symbols appearing in it, provided that the complexity of the symbols Type and $\emptyset$ is 1, the complexity of a constant/variable is 2, the complexity of the symbol $\mathcal{U}$ is greater than the complexity of $\mathcal{L}$, and the complexity of the subject of the judgement is the sum of the complexities of its symbols plus the complexity of the normal form of the type of each subterm of the subject, derived in the given context and signature.

1. Directly, using the induction hypothesis.

2. Directly, using the induction hypothesis.

3. Directly, using the induction hypothesis.

4. If $\Gamma \vdash_\Sigma \alpha : K$ is in η-lnf, we have that $\alpha = aN_1 \dots N_n$, $K \equiv \mathsf{Type}$, and also that:

   (a) $a : \Pi x_1{:}\sigma_1 \dots x_n{:}\sigma_n.\mathsf{Type} \in \Sigma$, with $\Pi x_1{:}\sigma_1 \dots x_n{:}\sigma_n.\mathsf{Type}$ in η-lnf. This means that, for all $1 \le i \le n$, we have that σ_i is in η-lnf.

   (b) $\Gamma \vdash_\Sigma N_i : \sigma'_i$ is derivable in $\mathsf{LF}_\mathcal{P}$, for $1 \le i \le n$, where σ'_i is in η-lnf, and $\sigma'_i =_{\beta\mathcal{L}} \sigma_i[N_1/x_1, \dots, N_{i-1}/x_{i-1}]$.

Now, from (a), we obtain that $\Gamma \vdash_\Sigma a \Rightarrow \Pi x_1{:}\sigma_1 \ldots x_n{:}\sigma_n.\mathsf{Type}$ is derivable in $\mathsf{CLF}_\mathcal{P}$. Next, by applying the induction hypothesis to (b), we obtain that $\Gamma \vdash_\Sigma N_i \Leftarrow \sigma_i'$ is derivable in CLF for all i. From this, by repeatedly applying the rule ($A{\cdot}App$), we get that $\Gamma \vdash_\Sigma aN_1 \ldots N_n \Rightarrow Type$.

5. The cases when σ is an atomic family have already been covered above, while the remaining cases are proven directly, using the induction hypothesis.

6. (a) If $\Gamma \vdash_\Sigma c : \alpha$, this could have been obtained only through the rule ($O{\cdot}Const$) of $\mathsf{LF}_\mathcal{P}$, from $\vdash_\Sigma \Gamma$ and $c : \sigma \in \Gamma$. By the induction hypothesis, we get that $\vdash_\Sigma \Gamma$ in $\mathsf{CLF}_\mathcal{P}$, and that $c : \sigma \in \Gamma$, from which, using the rule ($O{\cdot}Const$) of $\mathsf{CLF}_\mathcal{P}$, we obtain the desired $\Gamma \vdash_\Sigma c \Rightarrow \sigma$.

   (b) If $\Gamma \vdash_\Sigma x : \sigma$, this could have been obtained only through the rule ($O{\cdot}Var$) of $\mathsf{LF}_\mathcal{P}$, from $\vdash_\Sigma \Gamma$ and $x : \sigma \in \Gamma$. By the induction hypothesis, we get that $\vdash_\Sigma \Gamma$ in $\mathsf{CLF}_\mathcal{P}$, and that $x : \sigma \in \Gamma$, from which, using the rule ($O{\cdot}Var$) of $\mathsf{CLF}_\mathcal{P}$, we obtain the desired $\Gamma \vdash_\Sigma x \Rightarrow \sigma$.

   (c) Let $\Gamma \vdash_\Sigma AM : \alpha$ be derivable in $\mathsf{LF}_\mathcal{P}$, and be in η-lnf. A, as an atomic object, is then of the form:
   $$\mathcal{U}^{\mathcal{P}_1}_{N_1,\sigma_1}[\ldots[\mathcal{U}^{\mathcal{P}_k}_{N_k,\sigma_k}[c|x\{M_1 \ldots M_n\}]\overrightarrow{M_k}]\overrightarrow{M_{k-1}}\ldots]\overrightarrow{M_1}.$$
   Here, we have that c (or x) is fully applied and unlocked with respect to its classifier, and we also have that all N_i, σ_i, M_i, $\overrightarrow{M_i}$, as well as M and α, are in η-lnf. Also, we have that the types of M_i, $\overrightarrow{M_i}$, and M are in η-lnf, as they are recorded in the type of c (or x), which is part of the signature (or context), which is also in η-lnf. We will denote the type of M by σ, and the type of A by $\Pi z{:}\sigma.\tau$. Then, by the induction hypothesis, we have that $\Gamma \vdash_\Sigma M \Leftarrow \sigma$, and that $\Gamma \vdash_\Sigma A \Leftarrow \Pi z{:}\sigma.\tau$. From the latter, as it only could have been obtained through the rule ($O{\cdot}Atom$), we have that $\Gamma \vdash_\Sigma A \Rightarrow \Pi z{:}\sigma.\tau$. Now, we have two cases to consider:

      i. $\alpha \equiv a$, a constant atomic type, which is the trivial case, as a is immune to both hereditary and standard substitution. We immediately get that $\tau \equiv a$, with which we can use the rule ($O{\cdot}App$) of $\mathsf{CLF}_\mathcal{P}$ to obtain the desired $\Gamma \vdash_\Sigma AM \Rightarrow \alpha$.

      ii. $\alpha \equiv aM_1 \ldots M_n$, a fully applied constant atomic type of arity n. Then, we have that $\tau \equiv aM_1' \ldots M_n'$. Due to the fact that M, and all M_i, M_i' are in normal form (as they are in η-lnf), we have that $\tau[M/z] \equiv a[M/z]M_1'[M/z]\ldots M_n'[M/z] \equiv aM_1 \ldots M_n \equiv \alpha$. However, due to the normal forms, the ordinary and hereditary substitution here coincide, yielding $\tau[M/z]^F_\sigma = \alpha$. With this, using the rule ($O{\cdot}App$), we obtain the desired $\Gamma \vdash_\Sigma AM \Rightarrow \alpha$.

   (d) Let us consider the case when $\Gamma \vdash_\Sigma \mathcal{U}^\mathcal{P}_{N,\sigma}[A] : \alpha'$. By inspection of the typing rules of $\mathsf{LF}_\mathcal{P}$, we have that the original introduction rule for our initial judgement had to have been $\Gamma \vdash_\Sigma \mathcal{U}^\mathcal{P}_{N,\sigma}[A] : \alpha$, derived from $\Gamma \vdash_\Sigma A : \mathcal{L}^\mathcal{P}_{N,\sigma}[\alpha]$, $\Gamma \vdash_\Sigma N : \sigma$, and $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$, where $\alpha{=}_{\beta\mathcal{L}}\alpha'$. By the induction hypothesis, we have $\Gamma \vdash_\Sigma N \Leftarrow \sigma$. Using the rule ($O{\cdot}Conv$) of $\mathsf{LF}_\mathcal{P}$, we can now get $\Gamma \vdash_\Sigma A : \mathcal{L}^\mathcal{P}_{N,\sigma}[\alpha']$ and from this, by the induction hypothesis, we get that $\Gamma \vdash_\Sigma A \Leftarrow \mathcal{L}^\mathcal{P}_{N,\sigma}[\alpha']$. Since that would only be possible through the rule ($O{\cdot}Atom$), we have that $\Gamma \vdash_\Sigma A \Rightarrow \mathcal{L}^\mathcal{P}_{N,\sigma}[\alpha']$ also holds. Finally, given the properties of the predicate induced in $\mathsf{CLF}_\mathcal{P}$ by the predicate $\mathcal{P}$ in $\mathsf{LF}_\mathcal{P}$, as stated in the formulation of the theorem, we can use the rule ($O{\cdot}Unlock$) of $\mathsf{CLF}_\mathcal{P}$ to obtain the desired $\Gamma \vdash_\Sigma \mathcal{U}^\mathcal{P}_{N,\sigma}[M] \Rightarrow \alpha'$.

7. (a) The cases when M is an atomic object have already been covered above.

   (b) Let us consider the case when $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \theta$. By inspection of the typing rules of $\mathsf{LF}_\mathcal{P}$, we have that $\theta \equiv \Pi x{:}\sigma'.\tau'$, where the original introduction rule for our initial judgement had to have been $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau$, derived from $\Gamma, x{:}\sigma \vdash_\Sigma M : \tau$, where $\sigma{=}_{\beta\mathcal{L}}\sigma'$, and $\tau{=}_{\beta\mathcal{L}}\tau'$. However, since σ, M, σ' and τ' are in η-lnf, they must also be in normal form, meaning that $\sigma \equiv \sigma'$, leaving us with $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau'$. Using the rule ($O{\cdot}Conv$) of $\mathsf{LF}_\mathcal{P}$, we can now get $\Gamma, x{:}\sigma \vdash_\Sigma M : \tau'$, and from this, by the induction hypothesis, we get that $\Gamma, x{:}\sigma \vdash_\Sigma M \Leftarrow \tau'$, from which, using the rule ($O{\cdot}Abs$) of $\mathsf{CLF}_\mathcal{P}$, we obtain the desired $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M \Leftarrow \Pi x{:}\sigma.\tau'$.

   (c) Let us consider the case when $\Gamma \vdash_\Sigma \mathcal{L}^\mathcal{P}_{N,\sigma}[M] : \theta$. By inspection of the typing rules of $\mathsf{LF}_\mathcal{P}$, we have that $\theta \equiv \mathcal{L}^\mathcal{P}_{N',\sigma'}[\rho']$, where the original introduction rule for our initial judgement had to have been $\Gamma \vdash_\Sigma \mathcal{L}^\mathcal{P}_{N,\sigma}[M] : \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho]$, derived from $\Gamma \vdash_\Sigma M : \rho$ and $\Gamma \vdash_\Sigma N : \sigma$, where $N{=}_{\beta\mathcal{L}}N'$,

$\sigma =_{\beta\mathcal{L}} \sigma'$, and $\rho =_{\beta\mathcal{L}} \rho'$. However, since M, N, σ, N', σ', and ρ' are in η-lnf, they must also be in normal form, meaning that $N \equiv N'$ and $\sigma \equiv \sigma'$, leaving us with $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]'$. By the induction hypothesis, we have $\Gamma \vdash_\Sigma N \Leftarrow \sigma$. Using the rule $(O{\cdot}Conv)$ of $\mathsf{LF}_{\mathcal{P}}$, we can now get $\Gamma \vdash_\Sigma M : \rho'$ and from this, by the induction hypothesis, we get that $\Gamma \vdash_\Sigma M \Leftarrow \rho'$. Finally, we can use the rule $(O{\cdot}Lock)$ of $\mathsf{CLF}_{\mathcal{P}}$ to obtain the desired $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \Leftarrow \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho']$.

□

Notice that, by the Correspondence Theorem above, any well-behaved predicate $\mathcal{P}$ in $\mathsf{LF}_{\mathcal{P}}$ induces a well-behaved predicate in $\mathsf{CLF}_{\mathcal{P}}$. Finally, notice that *not* all $\mathsf{LF}_{\mathcal{P}}$ judgements have a corresponding η-lnf. 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, as the rule $(O{\cdot}Unlock)$ can be applied only when the predicate holds.

6 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, or the structure of the derivation itself, or other structures and mechanisms which are not available at the object level. This is the case for substructural and program logics [1, 2, 12].

We have isolated a *library* of predicates on proof terms, whose patterns frequently occur in the examples. The main archetype is: "given constants or variables only occur with some modality $\mathcal{D}$ in subterms satisfying the decidable property $\mathcal{C}$". Modalities can be anyone of such phrases: "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 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 [18], we present a basic library of auxiliary functions, which can be used to introduce external predicates of the above archetypes.

We start by giving, yet another, encoding of the well known case of untyped λ-calculus, with a call-by-value evaluation strategy. This allows us to illustrate yet another paradigm for dealing with free and bound variables appropriate for $\mathsf{LF}_{\mathcal{P}}$. Then, we add on top of such λ-calculus an extension, in order to model a sort of *design by contract* functional language. Next, we discuss Modal Logics, both in Hilbert and Natural Deduction style, and we give a sketch of how to encode the non-commutative linear logic introduced in [31]. Another example, on program logics *à la* Hoare, appears in Section 6.4. We state adequacy theorems, and, due to lack of space, here provide proofs for only some of them. For the sake of simplicity, in the following examples, we use the notation $\sigma \to \tau$ for $\Pi x{:}\sigma.\tau$ if $x \notin \mathsf{Fv}(\tau)$. Moreover, we will omit the type σ in $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]$, when σ is clear from the context. As far as comparisons with alternate encodings in LF, we refer the reader to the general comments which have been made in Section 4.

Finally, in the Adequacy Theorems, we will be using the notion of *judgement in η-long normal form* (η-lnf), which has been introduced in Definition 3.

6.1 The Untyped λ-calculus

6.1.1 Free and bound variables.

Consider the well-known untyped λ-calculus:

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

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

The abovementioned "bindable" variables must neither be confused with bound, nor with free variables. For instance, the λ-term x (in which the variable is free) will be encoded by means of the term $\vdash_\Sigma$`(free n):term` for a suitable (encoding of a) natural number `n` (see Definition 5 below). On the other hand, the λ-term $\lambda x.x$ (in which the variable is obviously bound) will be encoded by $\vdash_\Sigma$ `(lam λx:term.x)`. However, when we "open" the abstraction $\lambda x.M$, considering the body M, we will encode the latter as `x:term` $\vdash_\Sigma \epsilon_{\{x\}}$(`M`), where $\epsilon_{\{x\}}$ is the encoding function defined later in this section. In this case, x is a *bindable* variable.

Definition 5 ($\mathsf{LF}_\mathcal{P}$ signature Σ_λ for untyped λ-calculus)**.**

```
term : Type                      nat : Type                      0 : nat
   S : nat -> nat               free : nat -> term
 app : term -> term -> term      lam : (term -> term) -> term
```

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 λ-calculus, we put:

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

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

Theorem 8 (Adequacy of syntax)**.** *Let $\{x_i\}_{i\in\mathbb{N}\setminus\{0\}}$ be an enumeration of the variables in the λ-calculus. Then, the encoding function $\epsilon_\mathcal{X}$ is a bijection between the λ-calculus terms with bindable variables in $\mathcal{X}$ and the terms M derivable in judgements $\Gamma \vdash_{\Sigma_\lambda} M$: `term` in η-lnf, where $\Gamma = \{\texttt{x} : \texttt{term} \mid x \in \mathcal{X}\}$. Moreover, the encoding is compositional, i.e. for a term M, with bindable variables in $\mathcal{X} = \{x_1, \ldots, x_k\}$, and $N_1, \ldots, N_k$, with bindable variables in $\mathcal{Y}$, the following holds:*

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

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

$$\begin{aligned}\delta_\mathcal{X}((\texttt{free i})) &= x_i \text{ (where } x_i \notin \mathcal{X}) \\ \delta_\mathcal{X}(\texttt{xi}) &= x_i \text{ (where } x_i \in \mathcal{X}) \\ \delta_\mathcal{X}((\texttt{app M N})) &= \delta_\mathcal{X}(\texttt{M})\,\delta_\mathcal{X}(\texttt{N}) \\ \delta_\mathcal{X}(\texttt{lam}\ \lambda\texttt{x:term.M}) &= \lambda x.\delta_{\mathcal{X}\cup\{x\}}(\texttt{M})\end{aligned}$$

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

Definition 6 (The call-by-value reduction strategy)**.** The call-by-value (CBV) evaluation strategy can be specified by:

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

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

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

where values are either variables (constants in our encoding) or functions.

Definition 7 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ_{CBV} for λ-calculus CBV reduction)**.** We extend the signature of Definition 5 as follows:

```
   triple : Type
<_, _, _> : term -> (term -> term) -> (term -> term) -> triple
       eq : term -> term -> Type
     refl : ΠM:term.(eq M M)
     symm : ΠM:term.ΠN:term.(eq N M) -> (eq M N)
    trans : ΠM,N,P:term.(eq M N) -> (eq N P) -> (eq M P)
   eq_app : ΠM,N,M',N':term.(eq M N) -> (eq M' N') -> (eq (app M M') (app N N'))
    betav : ΠM:(term -> term).ΠN:term.L^Val_N[eq (app (lam M) N) (M N)]
     csiv : ΠM,N:(term -> term).Πx:term.L^ξ_⟨x,M,N⟩[(eq (M x)(N x))->(eq (lam M)(lam N))]
```

where `triple` is the type of triples of terms with types `term`, `term -> term` and `term -> term`, and the predicates *Val* and ξ are defined as follows:

- *Val*$(\Gamma \vdash_\Sigma$ `N : term`$)$ holds iff either `N` is an abstraction or a constant (*i.e.* a term of the shape `(free i)`);
- $\xi(\Gamma \vdash_\Sigma$ `<x,M,N> : 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 9 (Adequacy of CBV reduction)**.** *Given an enumeration $\{x_i\}_{i\in\mathbb{N}\setminus\{0\}}$ of the variables in the λ-calculus, there is a bijection between derivations of the judgment $\vdash_{CBV} M = N$ on terms with no bindable variables in the CBV λ-calculus and proof terms* `h`*, such that $\vdash_{\Sigma_{CBV}}$* `h` $:$ `(eq` $\epsilon_\emptyset(M)$ $\epsilon_\emptyset(N)$`)` *is in η-lnf.*

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

- If $\mathcal{D}$ is the derivation
$$\frac{}{\vdash_{CBV} M = M}$$
then $\epsilon^{=}_{\emptyset}(\mathcal{D}) =$ `(refl` $\epsilon_\emptyset(M)$`):(eq` $\epsilon_\emptyset(M)$ $\epsilon_\emptyset(M)$`)`.

- If $\mathcal{D}$ is the derivation with `(symm)` as last applied rule, then, by the induction hypothesis (IH), there is a term `h` such that $\vdash_{\Sigma_{CBV}}$ `h` $:$ `(eq` $\epsilon_\emptyset(N)$ $\epsilon_\emptyset(M)$`)`. Hence, we will have that $\epsilon^{=}_{\emptyset}(\mathcal{D}) =$ `(symm` $\epsilon_\emptyset(M)$ $\epsilon_\emptyset(N)$ `h):(eq` $\epsilon_\emptyset(M)$ $\epsilon_\emptyset(N)$`)`.

- If $\mathcal{D}$ is the derivation with `(trans)` as last applied rule, then, by the IH, we have that there exist terms `h` and `h'` such that $\vdash_{\Sigma_{CBV}}$ `h` $:$ `(eq` $\epsilon_\emptyset(M)$ $\epsilon_\emptyset(N)$`)`, and also $\vdash_{\Sigma_{CBV}}$ `h'` $:$ `(eq` $\epsilon_\emptyset(N)$ $\epsilon_\emptyset(P)$`)`. Hence, we will have that $\epsilon^{=}_{\emptyset}(\mathcal{D}) =$ `(trans` $\epsilon_\emptyset(M)$ $\epsilon_\emptyset(N)$ $\epsilon_\emptyset(P)$ `h h'):(eq` $\epsilon_\emptyset(M)$ $\epsilon_\emptyset(P)$`)`.

- If $\mathcal{D}$ is the derivation with `(eq_app)` as last applied rule, then, by the induction hypothesis, we have that there exist terms `h` and `h'` such that $\vdash_{\Sigma_{CBV}}$ `h` $:$ `(eq` $\epsilon_\emptyset(M)$ $\epsilon_\emptyset(N)$`)` and $\vdash_{\Sigma_{CBV}}$ `h'` $:$ `(eq` $\epsilon_\emptyset(M')$ $\epsilon_\emptyset(N')$`)`. Hence, we will have that $\epsilon^{=}_{\emptyset}(\mathcal{D}) =$ `(eq_app` $\epsilon_\emptyset(M)$ $\epsilon_\emptyset(N)$ $\epsilon_\emptyset(M')$ $\epsilon_\emptyset(N')$ `h h'):(eq (app` $\epsilon_\emptyset(M)$ $\epsilon_\emptyset(M')$`) (app` $\epsilon_\emptyset(N)$ $\epsilon_\emptyset(N')$`))`.

- If $\mathcal{D}$ is the derivation
$$\frac{v \text{ is a value}}{\vdash_{CBV} (\lambda x.M)v = M[v/x]}$$
then $\epsilon^{=}_{\emptyset}(\mathcal{D}) = \mathcal{U}^{Val}_{\epsilon_\emptyset(v)}$`[(betav (λx : term.`$\epsilon_{\{x\}}(M)$`)` $\epsilon_\emptyset(v)$`): (eq (app (lam λx : term.`$\epsilon_{\{x\}}(M)$`)` $\epsilon_\emptyset(v)$`) ((λx : term.`$\epsilon_{\{x\}}$ (M)`)(`$\epsilon_\emptyset(v)$`))))]`[3].

[3]Notice the presence of the unlock operator in front of the $\mathsf{LF}_{\mathcal{P}}$ encoding: this is possible thanks to the fact that we know, by hypothesis (e.g., the premise of the applied derivation rule), that v is a value. Indeed, since values in the object language are either variables or abstractions and we are deriving things from the empty context, in this case v must be an abstraction $\lambda y.N$ (otherwise it would be a free variable and the derivation context could not be empty). Thus, it will be encoded into a term of the form `(lam λy:term.`$\epsilon_{\{y\}}(N)$`)` and the predicate *Val* is defined to hold on such terms, see Definition 7, whence the predicate *Val* holds on $\vdash_{CBV} \epsilon_\emptyset(v)$ `: term`).

- If $\mathcal{D}$ is the derivation with (ξ_v) as the last applied rule, then, by the IH, there exists a term $\mathtt{h}$ for which it holds that $\vdash_{\Sigma_{CBV}} \mathtt{h} : (\mathtt{eq}\ \epsilon_\emptyset(M)\ \epsilon_\emptyset(N))$[4]. With this in place, we have the following: $\epsilon^{=}_{\emptyset}(\mathcal{D}) = (\mathcal{U}^{\xi}_{\mathtt{T},\mathtt{triple}}[(\mathtt{csiv}\ \lambda\mathtt{x{:}term}.\epsilon_{\{x\}}(M)\ \lambda\mathtt{x{:}term}.\epsilon_{\{x\}}(N)\ \epsilon_\emptyset(x))]\ \mathtt{h}) : (\mathtt{eq}\ (\mathtt{lam}\lambda\mathtt{x{:}term}.\epsilon_{\{x\}}(M))\ (\mathtt{lam}\ \lambda\mathtt{x{:}term}.\epsilon_{\{x\}}(N)))$, where $\mathtt{T}$ is the triple $\langle\epsilon_\emptyset(x), (\lambda\mathtt{x{:}term}.\epsilon_{\{x\}}(M)), (\lambda\mathtt{x{:}term}.\epsilon_{\{x\}}(N))\rangle$.

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

$$\frac{\delta_\emptyset(\mathtt{N}) \text{ is a value}}{\vdash_{CBV} \delta_\emptyset(\mathtt{lam}\ (\lambda\mathtt{x{:}term}.\mathtt{M'}))\delta_\emptyset(\mathtt{N}) = \delta_\emptyset((\lambda\mathtt{x{:}term}.\mathtt{M'})\ \mathtt{N})}$$

and since we have that $\delta_\emptyset((\mathtt{lam}\ (\lambda\mathtt{x{:}term}.\mathtt{M'}))) = \lambda x.\delta_{\{x\}}(\mathtt{M'})$ (see proof of Th. 8), that $\lambda x.\delta_{\{x\}}(\mathtt{M'})\delta_\emptyset(\mathtt{N}) = \delta_{\{x\}}(\mathtt{M'})[\delta_\emptyset(\mathtt{N})/x]$ (β-reduction in CBV λ-calculus), that $\delta_\emptyset((\lambda\mathtt{x{:}term}.\mathtt{M'})\ \mathtt{N}) = \delta_\emptyset(\mathtt{M'}[\mathtt{N}/\mathtt{x}])$ (β-reduction in $\mathsf{LF}_\mathcal{P}$) and that $\delta_{\{x\}}(\mathtt{M'})[\delta_\emptyset(\mathtt{N})/x] = \delta_\emptyset(\mathtt{M'}[\mathtt{N}/\mathtt{x}])$ (by induction on the structure of $\mathtt{M'}$), we are done. Therefore, it is easy to verify by induction on η-long normal forms that $\delta^{=}_{\emptyset}$ is well-defined and total. Similarly, we can prove that $\delta^{=}_{\emptyset}$ is the inverse of $\epsilon^{=}_{\emptyset}$, making $\epsilon^{=}_{\emptyset}$ a bijection. □

We conclude this section illustrating the expressive power of $\mathsf{LF}_\mathcal{P}$ by encoding a restricted η-rule, which generalizes the one originally suggested by Plotkin ([29]), *i.e.*, $\lambda x.xx = \lambda x.x(\lambda y.xy)$

$$\mathtt{eta}\ :\ \Pi\mathtt{M{:}(term\ ->\ term)}.\mathcal{L}^{\eta}_{\mathtt{M}}[\mathtt{(eq\ (lam\ M)\ (lam\ (}\lambda\mathtt{x{:}term.(M\ (lam\ }\lambda\mathtt{y{:}term.(app\ x\ y)))))))]$$

where the predicate $\eta(\Gamma \vdash_\Sigma \mathtt{M\ :\ (term->term)})$ holds if and only if the outermost abstracted variable of $\mathtt{M}$ occurs in functional position among the head variables.

6.2 Design by Contract in Functional Style

In this section, we extend the untyped call-by-value λ-calculus of the previous example to accommodate a minimal functional language supporting the *design by contract* paradigm (see, *e.g.*, [23]). More precisely, we will enrich the λ-calculus with the conditional expression *cond(C,M)*, whose intended semantics is that of checking that the constraint C applied to M (denoted by $C(M)$) holds. As we will see, such expressions can be used to validate the *entry input* and the *exit value* on a function application. (The syntax of conditions C is omitted, a typical example is given by predicates on natural number (in)equalities, as in the Hoare Logic example in Section 6.4. Informally, C must be "something" which can be evaluated to true or false, when applied to its argument.) The syntax of expressions is defined as follows:

$$M, N, \ldots ::= x \mid M\ N \mid \lambda x.M \mid \mathit{cond}(C, M),$$

The call-by-value (CBV) evaluation strategy can be then extended by adding to the rules of Definition 6 the following one:

$$\frac{C(v) \text{ holds and } v \text{ is a value}}{\vdash_{CBV} \mathit{cond}(C, v) = v}\ (\mathsf{cond})$$

Notice that conditional expressions are *first-class* values, *i.e.* they can be passed as arguments of a function. So far, we can encode pre- and post-conditions in our language as follows:

$$\mathit{cond}(Q,\ (M\ (\mathit{cond}(P,\ N))))$$

Indeed, the key idea is that the above expression will reduce only if the argument N were to "pass" the pre-condition P (*i.e.*, the *input contract*) and if the application result MN were to "pass" the post-condition Q (*i.e.*, the *output contract*).

[4]Notice that the object variable x occurring in M and N is represented by a constant ($\mathtt{(free\ k)}$, 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 $\mathtt{x}$ of the metalanguage of $\mathsf{LF}_\mathcal{P}$.

We build upon the previous case study, encoding free variables by means of constants (e.g., natural numbers) embedded into terms, while bindable and bound variables will be represented by metavariables of the metalanguage. In the next definition, we provide an $\mathsf{LF}_{\mathcal{P}}$ signature for our language, extending Definition 5 (`bool` will represent the type of boolean values `true` and `false`, *i.e.*, the result of evaluating conditions C on their arguments M).

Definition 8 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ for design by contract λ-calculus)**.**

```
  nat : Type                                term : Type                      bool : Type
    0 : nat                                    S : nat -> nat               free : nat -> term
 cond : (term -> bool) -> term -> term      app : term -> term -> term
  lam : (term -> term) -> term
```

In order to make clear the role played by the types and constructors so far introduced, we fully specify the encoding function $\epsilon_{\mathcal{X}}$, mapping terms of the source language into the corresponding terms of $\mathsf{LF}_{\mathcal{P}}$, where $\mathcal{X}$ denotes a set of bindable variables. Given an enumeration $\{x_i\}_{i\in\mathbb{N}\setminus\{0\}}$ of the variables in the source language, we have the following:

$$\begin{aligned}
\epsilon_{\mathcal{X}}(x_i) &= \begin{cases} \texttt{xi} & \text{if } x_i \in \mathcal{X} \\ (\texttt{free i}) & \text{if } x_i \notin \mathcal{X} \end{cases} \\
\epsilon_{\mathcal{X}}(\lambda x.M) &= (\texttt{lam } \lambda\texttt{x:term.}\epsilon_{\mathcal{X}\cup\{\texttt{x}\}}(M)) \\
\epsilon_{\mathcal{X}}(cond(C,M)) &= (\texttt{cond } \epsilon_{\mathcal{X}}(C)\ \epsilon_{\mathcal{X}}(M)), \\
\epsilon_{\mathcal{X}}(MN) &= (\texttt{app } \epsilon_{\mathcal{X}}(M)\,\epsilon_{\mathcal{X}}(N))
\end{aligned}$$

We now present the encoding of the CBV reduction of our source language encoded in $\mathsf{LF}_{\mathcal{P}}$. Obviously, all the rules stated in the previous case study (about untyped λ-calculus) remain unchanged. There is only the need to account for the new reduction rule involving the new constructor `cond`.

Definition 9 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ for design by contract λ-calculus CBV reduction)**.**
We extend the signature of Definition 7 by adding the following constant:

$$\texttt{condv : }\Pi\texttt{C:(term -> bool).}\Pi\texttt{M:term.}\mathcal{L}^{\textit{Eval}}_{\texttt{(C M)},\texttt{bool}}[\texttt{(eq (cond C M) M)}]$$

where the external predicate ***Eval*** is defined as follows:

- ***Eval***$(\Gamma \vdash_{\Sigma} \texttt{(C M):bool})$ holds iff `M` is a value (i.e., an abstraction or a constant), `C` and `M` are closed and the evaluation of the condition `C` on the term `M` holds.

The expressive power of external predicates is fully exploited in the above example. Of course, we require that the external logical conditions C (corresponding to object-level expressions `C`) allow the ***Eval*** predicate to satisfy the requirements of Definition 1, *i.e.*, that they induce a well-behaved $\beta\mathcal{L}$-reduction. We conclude by illustrating how the expressivity of predicates and locks allows for a straightforward encoding of the *design-by-contract* predecessor function as follows (taking `nat` as the type of terms):

$$\texttt{(lam } \lambda\texttt{x:nat.(cond (}\lambda\texttt{z:nat.z}\geq\texttt{ 0) ((cond (}\lambda\texttt{y:nat.y>0) x) - 1)))}$$

6.3 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 $\Box$-introduction rules in Modal logics (see, e.g., [1, 2, 12]). For the sake of readability, in the following we will often use an *infix* notation for encoding binary logic operators.

6.3.1 Modal Logics in Hilbert style.

In this example, we show how $\mathsf{LF}_{\mathcal{P}}$ permits 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 locked types in rules of proof and standard types in the rules of derivation.

By way of example, we give the signature (see Figure 12) for the classical Modal Logics K, KT, $K4$, $KT4$ (S_4), $KT45$ (S_5) in Hilbert style (see Figure 11); all feature necessitation (rule NEC in Figure 11) as a rule of proof. We make use of the predicate $\mathit{Closed}(\Gamma \vdash_\Sigma \mathtt{m}{:}\mathit{True}(\phi))$, which holds iff "all free variables occurring in `m` 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 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 (see [18] 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. Hence, 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 12).

$$
\begin{array}{lcl}
\mathsf{A}_1 & : & \phi \to (\psi \to \phi) \\
\mathsf{A}_2 & : & (\phi \to (\psi \to \xi)) \to (\phi \to \psi) \to (\phi \to \xi) \\
\mathsf{A}_3 & : & (\neg\phi \to \neg\psi) \to ((\neg\phi \to \psi) \to \phi) \\
\mathsf{K} & : & \Box(\phi \to \psi) \to (\Box\phi \to \Box\psi) \\
\top & : & \Box\phi \to \phi \\
\mathsf{4} & : & \Box\phi \to \Box\Box\phi \\
\mathsf{5} & : & \Diamond\phi \to \Box\Diamond\phi \\
\mathsf{MP} & : & \dfrac{\phi \quad \phi \to \psi}{\psi} \\
\mathsf{NEC} & : & \dfrac{\phi}{\Box\phi}
\end{array}
$$

C	$A_1 + A_2 + A_3 + MP$
K	$C + K + NEC$
KT	$K + T$
$K4$	$K + 4$
$KT4$	$KT + 4$
$KT45$	$KT4 + 5$

Figure 11: Hilbert style rules for Modal Logics

```
   o : Type        → : o -> o -> o        ¬ : o -> o        □ : o -> o        ◊ : o -> o
True : o -> Type
  A1 : Πϕ,ψ:o.True(ϕ→(ψ→ϕ))
  A2 : Πϕ,ψ,ξ:o.True(ϕ→(ψ→ξ))→(ϕ → ψ)→(ϕ → ξ))
  A3 : Πϕ,ψ:o.True((¬ϕ → ¬ψ)→((¬ϕ → ψ)→ ϕ))
   K : Πϕ,ψ:o.True(□(ϕ→ψ)→(□ϕ→□ψ))
   T : Πϕ:o.True(□ϕ→ϕ)
   4 : Πϕ:o.True(□ϕ→□□ϕ)
   5 : Πϕ:o.True(◊ϕ→□◊ϕ)
  MP : Πϕ,ψ:o.True(ϕ) -> True(ϕ→ψ) -> True(ψ)
 NEC : Πϕ:o.Πm:True(ϕ).ℒ_m^Closed[True(□ϕ)]
```

Figure 12: The signature Σ for the Modal Logics in Hilbert style

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

$$
\begin{array}{lcl}
\epsilon_{\mathcal{X}}(x) & = & \mathtt{x}, \text{ where } x \in \mathcal{X}; \\
\epsilon_{\mathcal{X}}(\neg\phi) & = & \neg\epsilon_{\mathcal{X}}(\phi); \\
\epsilon_{\mathcal{X}}(\phi \to \psi) & = & \epsilon_{\mathcal{X}}(\phi) \to \epsilon_{\mathcal{X}}(\psi); \\
\epsilon_{\mathcal{X}}(\Box\phi) & = & \Box\epsilon_{\mathcal{X}}(\phi). \\
\epsilon_{\mathcal{X}}(\Diamond\phi) & = & \Diamond\epsilon_{\mathcal{X}}(\phi).
\end{array}
$$

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

Theorem 10 (Adequacy of modal formulæ syntax)**.** *The encoding function $\epsilon_{\mathcal{X}}$ is a bijection between the Modal Logic formulæ with free variables in $\mathcal{X}$ and the terms ϕ derivable in judgements $\Gamma \vdash_{\Sigma_\Box} \phi : \mathtt{o}$ in η-lnf, where $\Gamma = \{\mathtt{x} : \mathtt{o} \mid x \in \mathcal{X}\}$. Moreover, the encoding is compositional, i.e. for a formula ϕ, with free variables in $\mathcal{X} = \{x_1, \ldots, x_k\}$, and $\psi_1, \ldots, \psi_k$, with free variables in $\mathcal{Y}$, the following holds: $\epsilon_{\mathcal{X}}(\phi[\psi_1, \ldots, \psi_k/x_1, \ldots, x_k]) = \epsilon_{\mathcal{X}}(\phi)[\epsilon_{\mathcal{Y}}(\psi_1), \ldots, \epsilon_{\mathcal{Y}}(\psi_k)/x_1, \ldots, x_k]$.*

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

Theorem 11 (Adequacy of the truth system in Hilbert-style)**.** *There is a bijection between derivations $\phi_1, \ldots, \phi_k \vdash \phi$ in Hilbert-style Modal Logic and proof terms $\mathtt{h}$ such that $\Gamma \vdash_\Sigma \mathtt{h} : (\mathtt{True}\ \epsilon_{\mathcal{X}}(\phi_1 \to \ldots \to \phi_k \to \phi))$ in η-long normal form, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is the set of propositional variables occurring in $\phi_1, \ldots, \phi_k, \phi$ and $\Gamma = \{\mathtt{x1} : \mathtt{o}, \ldots, \mathtt{xn} : \mathtt{o}\}$.*

6.3.2 Modal Logics in Natural Deduction 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. In particular there are several alternative formulations for S_4 and S_5, *e.g.* as defined by Prawitz, which have rules with rather elaborate restrictions on the shape of subformulae where assumptions occur. Figure 13 shows all the rules allowing to specify Modal Logics S_4 and S_5 (*à la* Prawitz), NK, NKT, NK4, NKT4, NKT45. In order to illustrate the flexibility of the system, the rule for $\Box$ introduction in S_4 ($(\Box\text{-}\mathsf{I}\cdot S_4)$) is given in the form which allows cut-elimination. Figure 14 shows their encoding in $\mathsf{LF}_{\mathcal{P}}$. Again, the crucial role is played by three predicates, namely *Closed*, *BoxedS4* and *BoxedS5*. As in the Hilbert-style encoding, *Closed*$(\Gamma \vdash_\Sigma \mathtt{m}{:}\mathit{True}(\phi))$ holds iff "all free variables occurring in $\mathtt{m}$ have type $\mathtt{o}$". This is precisely what is needed to correctly encode rule $(\Box'\text{-}\mathsf{I})$, where the truth of the formula ϕ must not depend on any assumptions.

In the case of the Modal Logic S_4, the intended meaning of *BoxedS4*$(\Gamma \vdash_\Sigma \mathtt{m}{:}\ \mathtt{True}(\phi))$ is that all 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 *BoxedS5*$(\Gamma \vdash_\Sigma \mathtt{m}{:}\ \mathtt{True}(\phi))$ holds iff the variables of $\mathtt{m}$ have type $\mathtt{True}(\Box\psi)$, $\mathtt{True}(\neg\Box\psi)$ or $\mathtt{o}$. 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 14).

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, 12]. In our approach, we avoid the explicit introduction in the encodings of extra-judgments and structures, as in [1, 2, 12], by delegating such machinery to an *external oracle* via locks.

As for the adequacy of our encoding, we can state Theorems 12 and 13 below. As in the previous case, we first define an encoding function $\epsilon_{\mathcal{X}}$ on formulæ with free variables in $\mathcal{X}$ as follows, representing atomic formulæ by means of $\mathsf{LF}_{\mathcal{P}}$ metavariables:

- $\epsilon_{\mathcal{X}}(x) = \mathtt{x}$, where $x \in \mathcal{X}$;
- $\epsilon_{\mathcal{X}}(\mathit{ff}) = \mathtt{ff}$;
- $\epsilon_{\mathcal{X}}(\neg\phi) = \neg\epsilon_{\mathcal{X}}(\phi)$;
- $\epsilon_{\mathcal{X}}(\phi \to \psi) = \epsilon_{\mathcal{X}}(\phi) \to \epsilon_{\mathcal{X}}(\psi)$;
- $\epsilon_{\mathcal{X}}(\Box\phi) = \Box\epsilon_{\mathcal{X}}(\phi)$;
- $\epsilon_{\mathcal{X}}(\Diamond\phi) = \Diamond\epsilon_{\mathcal{X}}(\phi)$;

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

Theorem 12 (Adequacy of modal formulæ syntax)**.** *The encoding function $\epsilon_{\mathcal{X}}$ is a bijection between the Modal Logic formulæ with free variables in $\mathcal{X}$ and the terms ϕ derivable in judgements $\Gamma \vdash_\Sigma \phi : \mathtt{o}$ in η-lnf, where $\Gamma = \{\mathtt{x} : \mathtt{o} \mid x \in \mathcal{X}\}$. Moreover, the encoding is compositional, i.e. for a formula ϕ, with free variables in $\mathcal{X} = \{x_1, \ldots, x_k\}$, and $\psi_1, \ldots, \psi_k$, with free variables in $\mathcal{Y}$, the following holds: $\epsilon_{\mathcal{X}}(\phi[\psi_1, \ldots, \psi_k/x_1, \ldots, x_k]) = \epsilon_{\mathcal{X}}(\phi)[\epsilon_{\mathcal{Y}}(\psi_1), \ldots, \epsilon_{\mathcal{Y}}(\psi_k)/x_1, \ldots, x_k]$.*

$$\frac{}{\Gamma, \phi \vdash \phi}\ (\mathsf{start}) \qquad \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}\ (\to -\mathsf{I}) \qquad \frac{\Gamma \vdash \phi \to \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}\ (\to -\mathsf{E})$$

$$\frac{\Gamma, \neg\phi \vdash \mathit{ff}}{\Gamma \vdash \phi}\ (\mathit{ff} - \mathsf{I}) \qquad \frac{\Gamma \vdash \mathit{ff}}{\Gamma \vdash \phi}\ (\mathit{ff} - \mathsf{E})$$

$$\frac{\Delta \vdash \Box\Gamma \quad \Box\Gamma \vdash \phi}{\Delta \vdash \Box\phi}\ (\Box - \mathsf{I} \cdot \mathsf{S_4}) \qquad \frac{\Box\Gamma_0, \neg\Box\Gamma_1 \vdash \phi}{\Box\Gamma_0, \neg\Box\Gamma_1 \vdash \Box\phi}\ (\Box - \mathsf{I} \cdot \mathsf{S_5}) \qquad \frac{\Gamma \vdash \Box\phi}{\Gamma \vdash \phi}\ (\Box - \mathsf{E})$$

$$\frac{\Gamma \vdash \Box(\phi \to \psi) \quad \Gamma \vdash \Box\phi}{\Gamma \vdash \Box\psi}\ (\to_\Box -\mathsf{E})$$

$$\frac{\emptyset \vdash \phi}{\emptyset \vdash \Box\phi}\ (\Box' - \mathsf{I}) \qquad \frac{\Gamma \vdash \Box\phi}{\Gamma \vdash \Box\Box\phi}\ (\Box_\Box - \mathsf{I}) \qquad \frac{\Gamma \vdash \Diamond\phi}{\Gamma \vdash \Box\Diamond\phi}\ (\Box_\Diamond - \mathsf{I})$$

NC	$\mathsf{start} + (\to -\mathsf{I}) + (\to -\mathsf{E}) + (\mathsf{RAA}) + (\mathit{ff} -\mathsf{I}) + (\mathit{ff} -\mathsf{E})$
S_4	$NC + (\Box - \mathsf{I} \cdot \mathsf{S_4}) + (\Box - \mathsf{E})$
S_5	$NC + (\Box - \mathsf{I} \cdot \mathsf{S_5}) + (\Box - \mathsf{E})$
NK	$NC + (\to_\Box -\mathsf{E}) + (\Box' - \mathsf{I})$
NKT	$NK + (\Box - \mathsf{E})$
$NK4$	$NK + (\Box_\Box - \mathsf{I})$
$NKT4$	$NKT + (\Box_\Box - \mathsf{I})$
$NKT45$	$NKT4 + (\Box_\Diamond - \mathsf{I})$

Figure 13: Modal Logic rules in Natural Deduction style

```
      o : Type   ff : o   ¬ : o -> o   → :  o -> o -> o     □ : o -> o   ◊ : o -> o
   True : o -> Type
   impI : Πϕ,ψ:o.(True(ϕ) -> True(ψ)) -> True(ϕ → ψ)
   impE : Πϕ,ψ:o.True(ϕ → ψ) -> True(ϕ) -> True(ψ)
    ffI : Πϕ:o.(True(¬ϕ) -> True(ff)) -> True(ϕ)
    ffE : Πϕ:o.True(ff) -> True(ϕ)
 BoxIS4 : Πϕ:o.Πm:True(ϕ).ℒ_m^BoxedS4[True(□ϕ)]
 BoxIS5 : Πϕ:o.Πm:True(ϕ).ℒ_m^BoxedS5[True(□ϕ)]
   BoxE : Πϕ:o.True(□ϕ) -> True(ϕ)
impBoxE : Πϕ,ψ:o.True(□(ϕ → ψ)) -> True(□ϕ) -> True(□ψ)
  BoxI' : Πϕ:o. Πm:True(ϕ).ℒ_m^Closed[True(□ϕ)]
BoxBoxI : Πϕ:o.True(□ϕ) -> True(□□ϕ)
BoxDiamondI : Πϕ:o.True(◊ϕ) -> True(□◊ϕ)
```

Figure 14: The signature Σ_S for classic S_4 Modal Logic in $\mathsf{LF}_\mathcal{P}$

The adequacy of the truth system of Modal Logics in natural deduction style can be proved by structural induction on derivations of the judgment $\Gamma \vdash \phi$:

Theorem 13 (Adequacy of Modal Logics in natural deduction style)**.** *Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of propositional variables occurring in formulæ $\phi_1, \ldots, \phi_k, \phi$. There exists a bijection between derivations of the judgement $\{\phi_1, \ldots, \phi_k\} \vdash \phi$ in Modal Logics in natural deduction style, and proof terms* `h` *such that $\Gamma \vdash_\Sigma$* `h` $:$ (`True` $\epsilon_\mathcal{X}(\phi)$) *in η-lnf, where $\Gamma = \{$*`x1` $:$ `o`, $\ldots,$ `xn` $:$ `o`, `h1` $:$ (`True` $\epsilon_\mathcal{X}(\phi_1)), \ldots,$ `hk` $:$ (`True` $\epsilon_\mathcal{X}$ $(\phi_k))\}$.*

6.3.3 Non-commutative linear logic (NCLL)

In this section, we outline an encoding in $\mathsf{LF}_{\mathcal{P}}$ of a substructural logic like the one presented in [31]. Take, for instance, the rules for the *ordered variables* and the $\twoheadrightarrow$ introduction/elimination rules[5]:

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

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

where the $\bowtie$ symbol denotes the *context merge operator*.

In this system "ordered assumptions occur exactly once and in the order they were made". In order to encode the condition on the occurrence of z as the last variable in the ordered context in the introduction rule, it is sufficient to make the observation that, in an LF-based logical framework, this information is fully recorded in the proof term. The last assumption made is the rightmost variable, the first is the leftmost. Therefore, we can, in $\mathsf{LF}_{\mathcal{P}}$, introduce suitable predicates in order to enforce such constraints, without resorting to complicated encodings. In the following, we present an encoding of this ordered fragment of NCLL into $\mathsf{LF}_{\mathcal{P}}$. In order to give a shallow encoding, we do not represent explicitly the proof terms of the original system (see, e.g., [31]). The encodings of rules $\twoheadrightarrow I$ and $\twoheadrightarrow E$ are:

```
impRightIntro : ΠA,B:o.ΠM:(True A)->(True B).L^{Rightmost}_{M,(True A)->(True B)}[(True (impRight A B))],
 impRightElim : ΠA,B:o.(True (impRight A B)) -> (True A) -> (True B),
```

where `True:o->Type` is the truth judgment on formulæ (represented by type `o`) and `impRight : o -> o -> o` represents the $\twoheadrightarrow$ constructor of *right ordered implications*. Finally, ***Rightmost***$(\Gamma \vdash_{\Sigma}$ `M:(True A)->(True B)`$)$ is the predicate checking that:

1. `M` is an abstraction (i.e., `M`$\equiv \lambda$`z : (True A).M'`);
2. all free variables in `M` occur in subterms whose type is either `o` or `(True A)` for some `A:o`;
3. the bound variable `z` occurs only once and neither to the right of a variable bound by an abstraction which is the third argument of `impRightIntro`, nor to the left of a variable bound by an abstraction which is the third argument of `impLeftIntro`, in the normal form of `M'`;
4. the bound variable `z` does not occur in the normal form of `M'`, in the fourth argument of the `impElim` and `impLinearElim` constructors.

As far as the $\rightarrowtail$ introduction/elimination rules, the encoding strategy is similar; indeed, the rules are the following:

$$\frac{\Gamma;\Delta;(z{:}A, \Omega) \vdash M{:}B}{\Gamma;\Delta;\Omega \vdash \lambda^{<}z{:}A.M{:}A \rightarrowtail B}\ (\rightarrowtail I)$$

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

Thus, our encoding exploits the predicate ***Leftmost*** in the encoding of the introduction rule of $\rightarrowtail$[6]:

```
impLeftIntro : ΠA,B:o.ΠM:(True A)->(True B).L^{Leftmost}_{M,(True A)->(True B)}[(True (impLeft A B))],
 impLeftElim : ΠA,B:o.(True A) -> (True (impLeft A B)) -> (True B),
```

where `impLeft : o -> o -> o` represents the $\rightarrowtail$ constructor of *left ordered implications*. Finally, ***Leftmost***$(\Gamma \vdash_{\Sigma}$ `M:(True A)->(True B)`$)$ checks the same constraints of the ***Rightmost*** predicate.

[5] Notice that in this logic the derivation context is split into three distinct parts, namely, the intuitionistic context Γ, the linear context Δ and the ordered context Ω.

[6] Notice that we switch the arguments of left ordered application in `impLeftElim`, in order to simplify the ordering constraints of the ***Rightmost*** and ***Leftmost*** predicates.

The fragment of linear implications is, again, treated in a similar way, with a suitable predicate "ensuring" the correct introduction of the $\multimap$ constructor. The rules for the linear fragment of NCLL are the following:

$$\frac{}{\Gamma; y{:}A; \cdot \vdash y{:}A}\ (lvar) \qquad \frac{\Gamma; (\Delta, y{:}A); \Omega \vdash M{:}B}{\Gamma; \Delta; \Omega \vdash \hat{\lambda} y{:}A.M{:}A \multimap B}\ (\multimap I)$$

$$\frac{\Gamma; \Delta_1; \Omega \vdash M{:}A \multimap B \quad \Gamma; \Delta_2; \cdot \vdash N{:}A}{\Gamma; (\Delta_1 \bowtie \Delta_2); \Omega \vdash M \hat{} N{:}B}\ (\multimap E)$$

Hence, the encodings of the rules $\multimap I$ and $\multimap E$ are as follows:

$$\texttt{impLinearIntro} : \Pi \texttt{A,B:o.}\Pi \texttt{M:(True A)->(True B).}\mathcal{L}^{Linear}_{\texttt{M,(True A)->(True B)}}[\texttt{(True (impLinear A B))}],$$
$$\texttt{impLinearElim} : \Pi \texttt{A,B:o.(True (impLinear A B)) -> (True A) -> (True B),}$$

where `impLinear : o -> o -> o` represents the $\multimap$ constructor of *linear implications*. Finally, we have $\mathit{Linear}(\Gamma \vdash_\Sigma \texttt{M:(True A)->(True B)})$ as the predicate checking that

1. `M` is an abstraction (*i.e.*, `M` $\equiv \lambda$`z:(True A).M'`);
2. all free variables in `M` occur in subterms whose type is either `o` or `(True A)` for some `A:o`;
3. the bound variable `z` occurs only once in the normal form of `M'`;
4. the bound variable `z` does not occur in the NF of `M'` in a subterm which is the fourth argument of the `impElim` constructor.

Finally, the encoding of the intuitionistic fragment of NCLL is straightforward, since in this part there are no restrictions about the intuitionistic variables:

$$\frac{}{(\Gamma_1, x{:}A, \Gamma_2); \cdot; \cdot \vdash x{:}A}\ (ivar) \qquad \frac{(\Gamma, x{:}A); \Delta; \Omega \vdash M{:}B}{\Gamma; \Delta; \Omega \vdash \lambda x{:}A.M{:}A \to B}\ (\to I)$$

$$\frac{\Gamma; \Delta; \Omega \vdash M{:}A \to B \quad \Gamma; \cdot; \cdot \vdash N{:}A}{\Gamma; \Delta; \Omega \vdash MN{:}B}\ (\to E)$$

The encodings of the introduction/elimination rules for the intuitionistic implication are trivial:

$$\texttt{impIntro} : \Pi \texttt{A,B:o.}\Pi \texttt{M:(True A)->(True B). (True (imp A B))},$$
$$\texttt{impElim} : \Pi \texttt{A,B:o.(True (imp A B)) -> (True A) -> (True B)}$$

Notice that in the encodings of rules $\twoheadrightarrow_E$, $\rightarrowtail E$ and $\multimap E$ we have not enforced any conditions on the free variables occurring in the terms, in order to avoid infringing the *closure under substitution* condition. Indeed, the obvious requirements surface in the adequacy theorem:

Theorem 14 (Adequacy for Non-Commutative Linear Logic)**.** *Let $\mathcal{X} = \{P_1, \ldots, P_n\}$ be a set of atomic formulæ occurring in formulæ $A_1, \ldots, A_k, A$. Then, there exists a bijection between derivations of the judgment $(A_1, \ldots, A_{i-1}); (A_i, \ldots, A_{j-1}); (A_j, \ldots, A_k) \vdash A$ in non-commutative linear logic, and proof terms* `h` *such that* $\Gamma_{\mathcal{X}}, \texttt{h}_1\texttt{:(True } \epsilon_{\mathcal{X}}(A_1)\texttt{)}, \ldots, \texttt{h}_k\texttt{:(True } \epsilon_{\mathcal{X}}(A_k)\texttt{)} \vdash \texttt{h} : \texttt{(True } \epsilon_{\mathcal{X}}\ (A)\texttt{)}$ *in η-long normal form, where the variables* $\texttt{h}_\texttt{i}, \ldots, \texttt{h}_{\texttt{j}-1}$ *occur in* `h` *only once (but never in the fourth argument of* `impElim`*),* $\texttt{h}_\texttt{j}, \ldots, \texttt{h}_\texttt{k}$ *occur in* `h` *only once and, precisely, in this order (but never in the fourth argument of* `impElim` *or* `impLinearElim`*), and $\Gamma_{\mathcal{X}}$ is the context* `P1:o,...,Pn:o` *representing the object-language propositional formulæ $P_1, \ldots, P_n$.*

As far as we know, this is the first example (see the discussion in, e.g., [12]) of an encoding of non-commutative linear logic in an LF-like framework. Notice the peculiarity of this adequacy result, which is inevitable given the substructural nature of NCLL, but which is, nonetheless, perfectly *compositional*. The gist is the following: as far as theorems, *i.e.* proofs with no assumptions, everything is standard; when assumptions, *i.e. truly free*, not *bindable* variables are involved, an external requirement has to be externally checked. An alternate adequacy could be stated representing, as in the λ-calculus case, truly free variables by constants. Obviously, carrying out a deep embedding of the system, one could enforce the conditions on the variables occurring in the linear and ordered contexts by means of suitable locks at the level of the proof terms (see, e.g., [18]).

6.4 Imp with Hoare Logic

In this subsection, we give an encoding of an imperative language, called Imp together with its Hoare Logic. The syntax of programs in Imp is:

$$\begin{array}{rcll} p & ::= & \mathit{skip} \mid x := \mathit{expr} \mid & \text{null} \mid \text{assignment} \\ & & \mathit{if\ cond\ then\ p\ else\ p} \mid p; p \mid & \text{cond} \mid \text{sequence} \\ & & \mathit{while\ cond}\ \{p\} & \text{while} \end{array}$$

Other primitive notions of our object language are variables, both integer and *identifier*, and expressions. Identifiers denote locations. For the sake of simplicity, we assume only integers (represented by type `int`) as possible values for identifiers. In this section, we follow as closely as possible the HOAS encoding, originally proposed in [1], in order to illustrate the features and possible advantages of using $\mathsf{LF}_{\mathcal{P}}$ w.r.t. LF. The main difference with that approach is that here we encode *concrete* identifiers by constants of type `var`, an `int`-like type, of course different from `int` itself, so as to avoid confusion with possible values of locations.

The syntax of variables and expressions is defined as follows:

Definition 10 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ for Imp)**.**

```
 int : Type                          bool : Type                          var  : Type
bang : var -> int                  0,1,-1 : int                 + : int -> int -> int
  =  : int -> int -> bool                              and, imp : bool -> bool -> bool
 not : bool -> bool                                     forall: (int -> bool) -> bool
```

Since variables of type `int` may be bound in expressions (by means of the `forall` constructor), we define explicitly the encoding function $\epsilon^{exp}_{\mathcal{X}}$ mapping expressions with bindable variables of type `int` in $\mathcal{X}$ of the source language Imp into the corresponding terms of $\mathsf{LF}_{\mathcal{P}}$:

$$\begin{array}{rcl} \epsilon^{exp}_{\mathcal{X}}(0) = \mathtt{0}, \quad \epsilon^{exp}_{\mathcal{X}}(1) & = & \mathtt{1}, \quad \epsilon^{exp}_{\mathcal{X}}(-1) = \mathtt{-1} \\ \epsilon^{exp}_{\mathcal{X}}(x) & = & \begin{cases} \mathtt{x} \text{ if } x \in \mathcal{X} \\ (\mathtt{bang\ x}) \text{ if } x \notin \mathcal{X} \end{cases} \\ \epsilon^{exp}_{\mathcal{X}}(n+m) = (\mathtt{+}\ \epsilon^{exp}_{\mathcal{X}}(n)\, \epsilon^{exp}_{\mathcal{X}}(m)), & & \epsilon^{exp}_{\mathcal{X}}(n=m) = (\mathtt{=}\ \epsilon^{exp}_{\mathcal{X}}(n)\, \epsilon^{exp}_{\mathcal{X}}(m)) \\ \epsilon^{exp}_{\mathcal{X}}(\neg e) = (\mathtt{not}\ \epsilon^{exp}_{\mathcal{X}}(e)), & & \epsilon^{exp}_{\mathcal{X}}(e \wedge e') = (\mathtt{and}\ \epsilon^{exp}_{\mathcal{X}}(e)\, \epsilon^{exp}_{\mathcal{X}}(e')) \\ \epsilon^{exp}_{\mathcal{X}}(e \supseteq e') = (\mathtt{imp}\ \epsilon^{exp}_{\mathcal{X}}(e)\, \epsilon^{exp}_{\mathcal{X}}(e')), & & \epsilon^{exp}_{\mathcal{X}}(\forall x.\phi) = (\mathtt{forall}\ \lambda \mathtt{x{:}int}.\epsilon^{exp}_{\mathcal{X}\cup\{x\}}(\phi)) \end{array}$$

where `x` in `(bang x)` denotes the encoding of the concrete memory location (*i.e.*, a constant of type `var`) representing the (free) source language identifier x; the other case represents the bindable variable x rendered as a $\mathsf{LF}_{\mathcal{P}}$ metavariable `x` of type `int` in HOAS style. The syntax of imperative programs is defined as follows:

Definition 11 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ for Imp with command)**.**
We extend the signature of Definition 10 as follows:

```
prog : Type            Iskip : prog          Iseq : prog -> prog -> prog
Iset : var -> int -> prog                    Iif : Πe:bool.prog -> prog -> L^QF_{e, bool}[prog]
Iwhile : Πe:bool.prog -> L^QF_{e, bool}[prog]
```

where the predicate $\mathit{QF}(\Gamma \vdash_{\Sigma} \mathtt{e{:}bool})$ holds iff the formula `e` is *closed* and *quantifier free*, i.e., it does not contain the `forall` constructor. We can look at QF as a "good formation" predicate, ruling out *bad programs with invalid boolean expressions* by means of stuck terms.

The encoding function $\epsilon^{prog}_{\mathcal{X}}$ mapping programs with free variables in $\mathcal{X}$ of the source language Imp into the corresponding terms of $\mathsf{LF}_{\mathcal{P}}$ is defined as follows:

$\epsilon^{prog}_{\mathcal{X}}(\mathit{skip}) = \mathtt{Iskip}$
$\epsilon^{prog}_{\mathcal{X}}(x := e) = (\mathtt{Iset\ x}\ \epsilon^{exp}_{\mathcal{X}}(e))$
$\epsilon^{prog}_{\mathcal{X}}(p;\ p') = (\mathtt{Iseq}\ \epsilon^{prog}_{\mathcal{X}}(p)\ \epsilon^{prog}_{\mathcal{X}}(p'))$
$\epsilon^{prog}_{\mathcal{X}}(\mathit{if}\ \ e\ \ \mathit{then}\ \ p\ \ \mathit{else}\ \ p') = \mathcal{U}^{\mathit{QF}}_{\epsilon^{exp}_{\mathcal{X}}(e),\mathtt{bool}}[(\mathtt{Iif}\ \epsilon^{\mathtt{exp}}_{\mathcal{X}}(\mathtt{e})\ \epsilon^{\mathtt{prog}}_{\mathcal{X}}(\mathtt{p})\ \epsilon^{\mathtt{prog}}_{\mathcal{X}}(\mathtt{p'}))]$ (*)
$\epsilon^{prog}_{\mathcal{X}}(\mathit{while}\ \ e\ \ \{p\}) = \mathcal{U}^{\mathit{QF}}_{\epsilon^{exp}_{\mathcal{X}}(e),\mathtt{bool}}[(\mathtt{Iwhile}\ \epsilon^{\mathtt{exp}}_{\mathcal{X}}(\mathtt{e})\ \epsilon^{\mathtt{prog}}_{\mathcal{X}}(\mathtt{p}))]$ (*)

(*) if e is a quantifier-free formula (we are assuming to encode *legal* programs).
Now, given the predicate `true: bool -> Type` such that `(true e)` holds iff `e` is true[7], we can define a judgment `hoare` as follows:

Definition 12 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ for Hoare)**.**

```
args : Type
     <_ , _> : var -> (int -> bool) -> args
       hoare : bool -> prog -> bool -> Type
 hoare_Iskip : Πe:bool.(hoare e Iskip e)
  hoare_Iset : Πt:int.Πx:var.Πe:int -> bool.
                                      L^{P^set}_{⟨x,e⟩, args}[(hoare (e t) (Iset x t) (e (bang x))]
  hoare_Iseq : Πe,e',e'':bool.Πp,p': prog.(hoare e p e') -> (hoare e' p' e'') ->
                                                          (hoare e (Iseq p p') e'')
   hoare_Iif : Πe,e',b:bool.Πp,p':prog.(hoare (b and e) p e') ->
                       (hoare ((not b) and e) p' e') -> (hoare e U^{QF}_{b,bool}[(Iif b p p')] e')
hoare_Iwhile : Πe,b:bool.Πp:prog.(hoare (e and b) p e) ->
                                        (hoare e U^{QF}_{b,bool}[(Iwhile b p)] ((not b) and e))
 hoare_Icons : Πe,e',f,f':bool.Πp:prog.(true (imp e' e)) -> (hoare e p f) ->
                                                  (true (imp f f')) -> (hoare e' p f'),
```

where $P^{set}(\Gamma \vdash_{\Sigma} \langle \mathtt{x},\mathtt{e}\rangle{:}\mathtt{args})$ holds iff `e` is closed[8] and the location (*i.e.*, constant) `x` does not occur in `e`. Such requirements amount to formalizing that no assignment made to the location denoted by `x` affects the meaning or value of `e` (*non-interference* property).

The intuitive idea here is that if $\mathtt{e}{=}\epsilon_{\mathcal{X}}^{exp}(E)$, $\mathtt{p}{=}\epsilon_{\mathcal{X}}^{prog}(P)$ and $\mathtt{e'}{=}\epsilon_{\mathcal{X}}^{exp}(E')$, then `(hoare e p e')` holds iff the Hoare's triple $\{E\}P\{E'\}$ holds. The advantage w.r.t. previous encodings (see, e.g., [1]), is that in $\mathsf{LF}_{\mathcal{P}}$ we can delegate to the external predicates `QF` and P^{set} all the complicated and tedious checks concerning *non-interference* of variables and good formation clauses for guards in the conditional and looping statements. Thus, the use of lock types, which are subject to the verification of such conditions, allows to legally derive $\Gamma \vdash_{\Sigma} \mathtt{m{:}(hoare\ e\ p\ e')}$ only according to the Hoare semantics.

7 A Philosophical Coda

One may still wonder whether it is worth to develop all the meta-theory we presented in this work. Why would one like to delegate part of the formalization and verification of an object system to an external tool? Indeed, one could adopt a "monolithic" approach, encoding and checking everything into his favourite proof assistant. However, such strategy will soon reveal many drawbacks.

We start by giving another practical application of Poincaré's Principle. Suppose we are in the setting of the π-calculus formalization (see, *e.g.*, [20]) and we want to prove rigorously that if a certain property $\mathcal{P}$ holds on a process P, then the same property will also hold on the process $(\nu x)P$, provided that $x \notin fn(P)$. With pencil and paper it is easy to conclude this, since we know that $P \equiv (\nu x)P$ under the aforementioned freshness assumption. However, proof assistants force the user to spell out in full details even trivial proofs like $P \equiv (\nu x)P$. Hence, if we want to formally prove the previous structural congruence statement, we have to proceed applying basic axioms:

$$\begin{array}{lcll} P|0 & \equiv & P & 0 \text{ is the identity w.r.t. parallel composition} \\ P|(\nu x)0 & \equiv & P & (\nu x)0 \equiv 0 \\ (\nu x)(P|0) & \equiv & P & (\nu x)(P|Q) \equiv P|(\nu x)Q \text{ if } x \notin fn(P) \\ (\nu x)P & \equiv & P & 0 \text{ is the identity w.r.t. parallel composition} \end{array}$$

Here, we showed only the main steps of the proof, using implicitly structural rules, transitivity and symmetry. Surely, this is rather boring and does not require any particular skill to carry it out: it is a trivial *verification* like checking, *e.g.*, that $2 + 2 = 4$ ([5, 30]). Of course, one will do the proof once

[7] The definition is omitted due to lack of space.
[8] Otherwise, the predicate P would not be well-behaved, see Definition 1.

and then save it for future reuse; however, at the level of the machinery behind the proof assistant, this would imply larger proof terms, more memory consumption and, in general, a slowdown in the overall performance[9]. Thus, being able to delegate such straightforward and trivial verifications to an external (and optimized) tool would be very helpful during complicated formal proof developments. Indeed, the user of the proof assistant would be free to concentrate only on the "creative" part of the whole proof and the framework itself would be free to avoid an explicit treatment of uninteresting parts of the proof. This is precisely an important aspect of the spirit behind the design of $\mathsf{LF}_{\mathcal{P}}$: allowing the user to factorize apart consolidated proof knowledge, freeing himself and the framework to record in full details "useless" and trivial verifications.

For instance, in the case of the π-calculus, the reduction rule taking into account structural congruences between processes, namely

$$\frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}$$

can be encoded in $\mathsf{LF}_{\mathcal{P}}$ using locks as follows:

$$\mathcal{L}^{\mathit{Struct}}_{\langle \mathtt{P},\mathtt{P}',\mathtt{Q}',\mathtt{Q}\rangle}[(\mathtt{red\ P\ Q})]$$

where the `red` symbol serves to encode the reduction relation $\longrightarrow$, and the *external* predicate ***Struct*** holds if and only if $P \equiv P'$ and $Q' \equiv Q$.

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

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

in *Deduction Modulo* can be represented as:

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

But the mechanism of lock/unlock types in $\mathsf{LF}_{\mathcal{P}}$ is more general than the one provided by Deduction Modulo or Poincare's principle. The latter can be viewed as extensions of the type Equality Rule to new definitional equalities. $\mathsf{LF}_{\mathcal{P}}$ on the other hand, allows to reflect these mechanisms into the proof objects themselves, as has been extensively shown in the examples.

8 Conclusions and Future Work

In this paper, we have presented an extension of the Edinburgh LF, which allows to internalize external validation tools and 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 [18], 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 simply need to extend the library of predicates [18].

In $\mathsf{LF}_{\mathcal{P}}$ we could address formally the issue of reflection. We can already grasp the gist of this philosophy through the following principle:

$$\mathtt{Reflection\ :\ \Pi x{:}o.}\ \mathcal{L}^{\mathit{isTrue}}_{x,o}[(\mathit{True}\ x)]$$

We believe that our framework could 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

[9] As we already pointed out in the introduction, proof terms can be kept small, adopting a reflection approach (see, e.g., [9]), but at the price of proving internally the correctness of the decision procedures.

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* ([24]).

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.

It should be noted that the system we have presented here is a purely first-order predicative type theory, corresponding to the vertex (1,0,0) of Barendregt's cube [4]. A reasonable and worthwhile step further would be to extend it to the full impredicative higher-order Calculus of Constructions.

Alternative presentations of $\mathsf{LF}_{\mathcal{P}}$ could be given, featuring say typed reductions, or doing away with unlock destructors. In certain cases, we might even want to keep track of the external calls which have been made during the derivation. One way to accomplish this would be to employ a variant of $\mathsf{LF}_{\mathcal{P}}$ in which $\mathcal{L}$-reductions do not fire, thus preserving the $\mathcal{U}$-$\mathcal{L}$ pairs within the term. More practical experimenting with $\mathsf{LF}_{\mathcal{P}}$ will provide more insights on these issues.

The prototype of $\mathsf{LF}_{\mathcal{P}}$ under development is about to be completed. 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.

Lastly, we believe that $\mathsf{LF}_{\mathcal{P}}$ can shed light on various concepts in need of better formal understanding, such as shallow vs. deep encodings, proposition-as-types for modal operators, and combinations of different validation tools.

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. Barendregt. Lambda Calculi with Types. In *Handbook of Logic in Computer Science*, volume II, pages 118–310. Oxford University Press, 1992.

[5] H.P. Barendregt and E. Barendsen. Autarkic computations in formal proofs. *Journal of Automated Reasoning*, 28:321–336, 2002.

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

[7] 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.

[8] L. Carroll. What the Tortoise Said to Achilles. *Mind*, 4:278–280, 1895.

[9] A. Chlipala. Library reflection. `http://adam.chlipala.net/cpdt/html/Reflection.html`.

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

[11] D. Cousineau and G. Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In Simona Ronchi Rocca, editor, *Typed Lambda Calculi and Applications*, volume 4583 of *Lecture Notes in Computer Science*, pages 102–117. Springer Berlin Heidelberg, 2007.

[12] K. Crary. Higher-order representation of substructural logics. In *ICFP '10*, pages 131–142. ACM, 2010.

[13] N. de Bruijn. Automath, a language for mathematics. In *Automation and Reasoning, volume 2, Classical papers on computational logic 1967-1970*, pages 159–200. Springer Verlag, 1968.

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

[15] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40:143–184, January 1993.

[16] R. Harper and D. Licata. Mechanizing metatheory in a logical framework. *J. Funct. Program.*, 17:613–673, 2007.

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

[18] F. Honsell, M. Lenisa, L. Liquori, P. Maksimovic, and I. Scagnetto. LF_P – a logical framework with external predicates. Technical report, Università di Udine, Italy, 2012, `www.dimi.uniud.it/scagnetto/Papers/lfp-full.pdf`.

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

[20] F. Honsell, M. Miculan, and I. Scagnetto. π-calculus in (Co)Inductive Type Theories. *Theoretical Computer Science*, 253(2):239–285, 2001.

[21] Furio Honsell, Marina Lenisa, Luigi Liquori, Petar Maksimovic, and Ivan Scagnetto. $\mathsf{LF}_{\mathcal{P}}$: a logical framework with external predicates. In *Proceedings of the seventh international workshop on Logical frameworks and meta-languages, theory and practice*, LFMTP '12, pages 13–22, New York, NY, USA, 2012. ACM.

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

[23] B. Meyer. Design by contract. In *Advances in Object-Oriented Software Engineering*, pages 1–50. Prentice Hall , Englewood Cliffs , NJ, 1991.

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

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

[26] F. Pfenning and C. Schuermann. Twelf user's guide, chapter 6: Constraint domains. Online at `http://www.cs.cmu.edu/~twelf/guide-1-4/twelf_6.html`. Accessed January 2013.

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

[28] B. Pientka and J. Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In *Automated Reasoning*, volume 6173 of *Lecture Notes in Computer Science*, pages 15–21. Springer Berlin / Heidelberg, 2010.

[29] G.D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(2):125 – 159, 1975.

[30] H. Poincaré. *La Science et l'Hypothèse*. Flammarion, Paris, 1902.

[31] J. Polakow and F. Pfenning. Natural deduction for intuitionistic non-commutative linear logic. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 295–309. Springer Berlin Heidelberg, 1999.

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

[33] A. Shopenhauer. *Il mondo come volontà e rappresentazione*. Newton Compton, 2011.

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

[35] R. Virga. *Higher-Order Rewriting with Dependent Types*. PhD thesis, Carnegie Mellon University, Pittsburgh, USA, 1999.

[36] K. Watkins, I. Cervesato, F. Pfenning, and D. Walker. A Concurrent Logical Framework I: Judgments and Properties. Tech. Rep. CMU-CS-02-101, School of Computer Science, Carnegie-Mellon University, Pittsburgh, 2002.

Appendix

Definitions and extensions of several standard notions

- the domain of a signature $\mathsf{Dom}(\Sigma)$:

$$\begin{aligned}\mathsf{Dom}(\emptyset) &= \emptyset\\ \mathsf{Dom}(\Sigma, a{:}K) &= \mathsf{Dom}(\Sigma) \cup \{a\}\\ \mathsf{Dom}(\Sigma, c{:}\sigma) &= \mathsf{Dom}(\Sigma) \cup \{c\}\end{aligned}$$

- the domain of a context $\mathsf{Dom}(\Gamma)$:

$$\begin{aligned}\mathsf{Dom}(\emptyset) &= \emptyset\\ \mathsf{Dom}(\Gamma, x{:}\sigma) &= \mathsf{Dom}(\Gamma) \cup \{x\}\end{aligned}$$

- the free variables of a term $\mathsf{Fv}(T)$:

$$\begin{aligned}\mathsf{Fv}(\mathsf{Type}) = \mathsf{Fv}(a) = \mathsf{Fv}(c) &= \emptyset,\\ \mathsf{Fv}(x) &= \{x\}\\ \mathsf{Fv}(\Pi x{:}\sigma.T) &= (\mathsf{Fv}(\sigma) \cup \mathsf{Fv}(T))\backslash\{x\}\\ \mathsf{Fv}(\lambda x{:}\sigma.T) &= (\mathsf{Fv}(\sigma) \cup \mathsf{Fv}(T))\backslash\{x\}\\ \mathsf{Fv}(T\ N) &= \mathsf{Fv}(T) \cup \mathsf{Fv}(N)\\ \mathsf{Fv}(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[T]) &= \mathsf{Fv}(N) \cup \mathsf{Fv}(\sigma) \cup \mathsf{Fv}(T)\\ \mathsf{Fv}(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[T]) &= \mathsf{Fv}(N) \cup \mathsf{Fv}(\sigma) \cup \mathsf{Fv}(T)\end{aligned}$$

- the bound variables of a term $\mathsf{Bv}(T)$:

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

- substitution $T[M/x]$ of an object M for the variable x in a term T (here, we assume that $x \neq y$, and that we are working modulo Barendregt's hygiene condition):

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

- substitution $T[M/x]$ of an object M for the variable x in a context Γ:

$$\begin{aligned}(\emptyset)[M/x] &= \emptyset\\ (\Gamma, y{:}\sigma)[M/x] &= \Gamma[M/x], y{:}(\sigma[M/x]),\ \text{where } x \neq y, \mathsf{Fv}(M) \subseteq \mathsf{Dom}(\Gamma).\end{aligned}$$

$\beta\mathcal{L}$-closure under context for kinds

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

Figure 15: $\beta\mathcal{L}$-closure-under-context for kinds

$\beta\mathcal{L}$-closure under context for objects

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

$$\frac{M \to_{\beta\mathcal{L}} M'}{M\,N \to_{\beta\mathcal{L}} M'\,N}\ (O{\cdot}A_1{\cdot}\beta\mathcal{L}) \qquad \frac{N \to_{\beta\mathcal{L}} N'}{M\,N \to_{\beta\mathcal{L}} M\,N'}\ (O{\cdot}A_2{\cdot}\beta\mathcal{L})$$

$$\frac{N \to_{\beta\mathcal{L}} N'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N',\sigma}[M]}(O{\cdot}\mathcal{L}_1{\cdot}\beta\mathcal{L}) \qquad \frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[M]}(O{\cdot}\mathcal{L}_2{\cdot}\beta\mathcal{L})$$

$$\frac{M \to_{\beta\mathcal{L}} M'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M']}(O{\cdot}\mathcal{L}_3{\cdot}\beta\mathcal{L}) \qquad \frac{N \to_{\beta\mathcal{L}} N'}{\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N',\sigma}[M]}(O{\cdot}\mathcal{U}_1{\cdot}\beta\mathcal{L})$$

$$\frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N,\sigma'}[M]}(O{\cdot}\mathcal{U}_1{\cdot}\beta\mathcal{L}) \qquad \frac{M \to_{\beta\mathcal{L}} M'}{\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M']}(O{\cdot}\mathcal{U}_1{\cdot}\beta\mathcal{L})$$

Figure 16: $\beta\mathcal{L}$-closure-under-context for objects of $\mathsf{LF}_{\mathcal{P}}$

$\beta\mathcal{L}$-definitional equality in $\mathsf{LF}_{\mathcal{P}}$

$$\frac{T \to_{\beta\mathcal{L}} T'}{T =_{\beta\mathcal{L}} T'}\ (\beta\mathcal{L}{\cdot}Eq{\cdot}Main) \qquad \frac{}{T =_{\beta\mathcal{L}} T}\ (\beta\mathcal{L}{\cdot}Eq{\cdot}Refl)$$

$$\frac{T =_{\beta\mathcal{L}} T'}{T' =_{\beta\mathcal{L}} T}\ (\beta\mathcal{L}{\cdot}Eq{\cdot}Sym) \qquad \frac{T =_{\beta\mathcal{L}} T' \qquad T' =_{\beta\mathcal{L}} T''}{T =_{\beta\mathcal{L}} T''}\ (\beta\mathcal{L}{\cdot}Eq{\cdot}Trans)$$

Figure 17: $\beta\mathcal{L}$-definitional equality in $\mathsf{LF}_{\mathcal{P}}$