Gluing together Proof Environments: Canonical extensions of LF Type Theories featuring Locks
Furio Honsell, Luigi Liquori, Petar Maksimovic, Ivan Scagnetto

To cite this version:
Furio Honsell, Luigi Liquori, Petar Maksimovic, Ivan Scagnetto. Gluing together Proof Environments: Canonical extensions of LF Type Theories featuring Locks . LFMTP’15. 9th International Workshop on Logical Frameworks and Meta-languages, Berlin, Germany, Aug 2015, Berlin, Germany. Electronic Proceedings in Theoretical Computer Science (EPTCS), 2015, <10.4204/EPTCS.185.1>. <hal-01170029>
Gluing together Proof Environments: Canonical extensions of LF Type Theories featuring Locks

Furio Honsell
Department of Mathematics and Computer Science
University of Udine, Italy
furio.honsell@uniud.it

Luigi Liquori
Inria Sophia Antipolis Méditerranée, France
luigi.liquori@inria.fr

Petar Maksimović
Inria Rennes Bretagne Atlantique, France
Mathematical Institute of the Serbian Academy of Sciences and Arts, Serbia
petar.maksimovic@inria.fr

Ivan Scagnetto
Department of Mathematics and Computer Science
University of Udine, Italy
ivan.scagnetto@uniud.it

We present two extensions of the LF Constructive Type Theory featuring monadic locks. A lock is a monadic type construct that captures the effect of an external call to an oracle. Such calls are the basic tool for gluing together diverse Type Theories and proof development environments. The oracle can either be invoked in order to check that a constraint holds or to provide a suitable witness. The systems are presented in the canonical style developed by the CMU School. The first system, CLLF, is the canonical version of the system LLF, presented earlier by the authors. The second system, CLLF?, features the possibility of invoking the oracle to obtain a witness satisfying a given constraint. We discuss encodings of Fitch-Prawitz Set theory, call-by-value λ-calculi, and systems of Light Linear Logic. Finally, we show how to use Fitch-Prawitz Set Theory to define a type system that types precisely the strongly normalizing terms.

1 Introduction

In recent years, the authors have introduced in a series of papers [17, 15, 20, 19] various extensions of the Constructive Type Theory LF, with the goal of defining a simple Universal Meta-language that can support the effect of gluing together, i.e. interconnecting, different type systems and proof development environments.

The basic idea underpinning these logical frameworks is to allow for the user to express explicitly, in an LF type-theoretic framework the invocation, and uniform recording of the effect, of external tools by means of a new monadic type-constructor $L_{\mathcal{M}\sigma}^\rho$, called a lock. More specifically, locks permit to express the fact that, in order to obtain a term of a given type, it is necessary to verify, first, a constraint $\mathcal{P}(\Gamma \vdash \Sigma \ M : \sigma)$, i.e. to produce suitable evidence. No restrictions are enforced on producing such evidence. It can be supplied by calling an external proof search tool or an external oracle, or exploiting some other epistemic source, such as diagrams, physical analogies, or explicit computations according to the Poincaré Principle [3]. Thus, by using lock constructors, one can factor-out the goal, produce pieces of evidence using different proof environments and glue them back together, using the unlock operator, which releases the locked term in the calling framework.

*The work presented in this paper was partially supported by the Serbian Ministry of Education, Science, and Technological Development, projects ON174026 and III44006.
One of the original contributions of this paper is that we show how locks can delegate to external tools not only the task of producing suitable evidence but also that of exhibiting suitable witnesses, to be further used in the calling environment.

Locks subsume different proof attitudes, such as proof-irrelevant approaches, where one is only interested in knowing that a proof witness exists, or approaches relying on powerful terminating metalanguages. Indeed, they allow for a straightforward accommodation of many different proof cultures within a single Logical Framework; which otherwise can be embedded only very deeply \cite{5,14} or axiomatically \cite{21}.

Differently from our earlier work, we focus in this paper only on systems presented in the canonical format introduced by the CMU school \cite{13} \cite{32}. This format is syntax-directed and produces a unique derivation for each derivable judgement. Terms are all in normal form and equality rules are replaced by hereditary substitution. Canonical systems are very handy in establishing adequacy of encodings.

First, we present the very expressive system CLLF_{\rho} and discuss the relationship to its non-canonical counterpart LLF_{\rho} in \cite{19}, where we introduced lock-types following the paradigm of Constructive Type Theory (à la Martin-Löf), via introduction, elimination, and equality rules. This paradigm needs to be rephrased for the canonical format used here. Introduction rules correspond to type checking rules of canonical objects, whereas elimination rules correspond to type synthesis rules of atomic objects. Equality rules are rendered via the rules of hereditary substitution. In particular, we introduce a lock constructor for building canonical objects \( \mathcal{L}_{N,\sigma}^{\rho}[M] \) of type \( \mathcal{L}_{N,\sigma}^{\rho}[\rho] \), via the type checking rule (O-Lock).

Correspondingly, we introduce an unlock destructor, \( \mathcal{U}_{N,\sigma}^{\rho}[M] \), and an atomic rule (O-Unlock), allowing elimination, in the hereditary substitution rules, of the lock-type constructor, under the condition that a specific predicate \( \varphi \) is verified, possibly externally, on a judgement:

\[
\frac{}{\Gamma \vdash \Sigma \mathcal{L}_{N,\sigma}^{\varphi}[M] \iff \mathcal{L}_{N,\sigma}^{\varphi}[\rho]} \quad (O-Lock) \\
\frac{}{\Gamma \vdash \Sigma \mathcal{U}_{N,\sigma}^{\rho}[A] \Rightarrow \mathcal{L}_{N,\sigma}^{\varphi}[\rho]} \quad (O-Unlock)
\]

Capitalizing on the monadic nature of the lock constructor, as we did for the systems in \cite{20,19}, one can replace locked terms without necessarily establishing the predicate. This increases the expressivity of the system, and allows for reasoning under the assumption that the verification is successful, as well as for postponing verifications. The crucial rules are:

\[
\frac{}{\Gamma, x: \tau \vdash \Sigma \mathcal{L}_{S,\sigma}^{\varphi}[\rho]} \quad (F-Nested-Unlock) \\
\frac{}{\Gamma \vdash \Sigma \mathcal{L}_{S,\sigma}^{\rho}[\rho']} \quad \rho \mathcal{U}_{S,\sigma}[A/x]^F_{(\tau)} = \rho' \\
\frac{}{\Gamma \vdash \Sigma \mathcal{U}_{S,\sigma}[A] \Rightarrow \mathcal{L}_{S,\sigma}^{\varphi}[\rho]} \quad \rho' \mathcal{L}_{S,\sigma}[A/x]^O_{(\tau)} = M' \\
\frac{}{\Gamma \vdash \Sigma \mathcal{L}_{S,\sigma}[M'] \iff \mathcal{L}_{S,\sigma}^{\varphi}[\rho]} \quad \text{(O-Nested-Unlock)}
\]

The (O-Nested-Unlock)-rule is the counterpart of the elimination rule for monads, once we realize that \( let_{\varphi(A)} x = A \) in \( N \) can be replaced by \( N[\mathcal{U}_{S,\sigma}[A]/x] \). And this holds since the \( \mathcal{L}_{S,\sigma}^{\varphi}[\cdot] \)-monad satisfies the property \( let_{\varphi} x = M \) in \( N \) if \( x \notin \text{Fv}(N) \), provided \( x \) occurs guarded in \( N \), i.e. within subterms of the appropriate lock-type. The rule (F-Nested-Unlock) takes care of elimination at the level of types.

We proceed then to introduce CLLF_{\rho?}. Syntactically, it might appear as a minor variation of CLLF_{\rho}, but the lock constructor is used here to express the request for a witness satisfying a given property, which is then replaced by the unlock operation. In CLLF_{\rho?}, the lock acts as a binding operator and the unlock as an application.
The judgements \( \Sigma \stackrel{\text{sig}}{\vdash} \Gamma \) is a valid signature, while \( \Gamma \vdash \Sigma K \) is \( K \) is a kind in \( \Gamma \) and \( \Sigma \). The judgements \( \Gamma \vdash \Sigma \sigma \) type is \( \sigma \) is a canonical family in \( \Gamma \) and \( \Sigma \). The judgements \( \Gamma \vdash \Sigma \alpha \Rightarrow K \) is the kind of the atomic family \( \alpha \) in \( \Gamma \) and \( \Sigma \). The judgements \( \Gamma \vdash \Sigma M \Leftarrow \sigma \) is a canonical term of type \( \sigma \) in \( \Gamma \) and \( \Sigma \). The judgements \( \Gamma \vdash \Sigma A \Rightarrow \sigma \) is the type of the atomic term \( A \) in \( \Gamma \) and \( \Sigma \).
Valid signatures

<table>
<thead>
<tr>
<th>$\emptyset$ sig</th>
<th>$\Sigma$ sig</th>
<th>$\Sigma K \not\in \text{Dom}(\Sigma)$</th>
<th>$\Sigma, a:K$ sig</th>
<th>$\Sigma, c:\sigma$ sig</th>
</tr>
</thead>
</table>

Kind rules

$\Gamma, \alpha: K \vdash K$ (K-Type)

$\Gamma, x: \sigma \vdash \Sigma, \Pi x: \sigma.K$ (K-Pi)

Atomic Family rules

$\vdash \Sigma, a: K \in \Sigma$ (A-Const)

$\Gamma, \alpha \in \Pi x: \sigma.K$ (A-Var)

$\Gamma, M \in \Sigma \sigma \vdash \sigma$ (A-Mapp)

Canonical Family rules

$\Gamma \vdash \Sigma, \alpha \Rightarrow \tau$ (F-Atom)

$\Gamma, x: \sigma \vdash \tau \Rightarrow \tau$ (F-Pi)

$\Gamma, \rho \Rightarrow \tau$ (F-Type)

$\Gamma, \Sigma, \rho \Rightarrow \sigma$ (F-Lock)

$\Gamma, x: \tau \Rightarrow \Sigma, \rho$ (F-Nested-Lock)

$\Gamma, \Sigma, \rho \Rightarrow \sigma$ (F-Unlock)

$\Gamma, \Sigma, \rho \Rightarrow \sigma$ (F-OpenLock)

$\Gamma, \Sigma, \rho \Rightarrow \sigma$ (F-OpenUnlock)

Figure 2: The CLLF $\beta$ Type System

$\Gamma \vdash \Sigma, M \in \sigma$. The type system makes use, in the rules (A-App) and (F-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 $T[M/x]_\rho = T'$, 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, $\rho$ is the simple-type of $M$, and $t$ denotes the syntactic class (e.g. atomic families/object, canonical families/objects, etc.) under consideration.

The simple-type $\rho$ of $M$ is obtained via the erasure function of $[13]$ (Figure 3), mapping dependent into simple-types. The rules for Hereditary Substitution are presented in Figures 4 and 5 using Barendregt’s hygiene condition.

Notice that, in the rule (O-Atom) of the type system (Figure 2), the syntactic restriction of the classifier to $\alpha$ atomic ensures that canonical forms are long $\beta\eta$-normal forms for the suitable notion of long
\(\frac{(a) = \rho}{(aM) = \rho}\)

\(\frac{(\sigma) = \rho_1 \quad (\tau) = \rho_2}{(\Pi x. \sigma. \tau) = \rho_1 \rightarrow \rho_2}\)

\(\frac{(\tau) = \rho}{(L_{N,\sigma}[\tau]) = L_{N,\sigma}[\rho]}\)

Figure 3: Erasure to simple-types

### Substitution in Kinds

\(\frac{\text{type}[M_0/x_0]^K_{\rho_0} = \text{type}}{\text{K-Type}}\)

\(\frac{\sigma[M_0/x_0]^K_{\rho_0} = \sigma' \quad K[M_0/x_0]^K_{\rho_0} = K'}{(\Pi x. \sigma. K)[M_0/x_0]^K_{\rho_0} = \Pi x. \sigma'. K'}\)

\(\frac{(\sigma \cdot M) = \sigma' M'}{(\sigma M)[M_0/x_0]^I_{\rho_0} = \sigma' M'}\)

\(\frac{\alpha[\sigma[\sigma_1 \cdots \sigma_n]]_{\rho_0} = \alpha' \quad M[\sigma[\sigma_1 \cdots \sigma_n]]_{\rho_0} = M'}{(\alpha M)[\sigma[\sigma_1 \cdots \sigma_n]]_{\rho_0} = \alpha' M'}\)

\(\frac{\alpha[\sigma_2][\sigma_1][\sigma_2][\sigma_1][\sigma_2]}{L_{M,\alpha}[\sigma_2][\sigma_1][\sigma_2]}\)

Figure 4: Hereditary substitution, kinds and families of CLLF_\varphi

\(\beta \eta\)-normal form, which extends the standard one for lock-types. For one, the judgement \(\Gamma \vdash \lambda x. a \cdot a \vdash \Sigma x \leftarrow \Sigma x \leftarrow \Sigma\) is not derivable, as \(\Pi \vdash a \cdot a\) is not atomic, hence \(\vdash \lambda x : (\Pi \vdash a \cdot a), x : \Pi \vdash x\). \(\Pi \vdash a \cdot a\) is not derivable. On the other hand, \(\vdash \lambda x : (\Pi \vdash a \cdot a), \lambda y : a \cdot x y \vdash \Pi \vdash x : (\Pi \vdash a \cdot a)\). \(\Pi \vdash a \cdot a\), where \(a\) is a family constant of kind Type, is derivable. Analogously, for lock-types, the judgement \(x : L_{N,\sigma}[\rho] \vdash \sigma \cdot x \leftarrow L_{N,\sigma}[\rho]\) is not derivable, since \(L_{N,\sigma}[\rho]\) is not atomic. As a consequence, we have that \(\vdash \lambda x : L_{N,\sigma}[\rho] \cdot x \leftarrow \Pi x : L_{N,\sigma}[\rho] \cdot L_{N,\sigma}[\rho]\) is not derivable. However, \(x : L_{N,\sigma}[\rho] \vdash \Sigma L_{N,\sigma}[\rho], L_{N,\sigma}[\rho] \leftarrow L_{N,\sigma}[\rho]\) is derivable, if \(\rho\) is atomic. Hence, the judgment \(\vdash \lambda x : L_{N,\sigma}[\rho], L_{N,\sigma}[\rho], L_{N,\sigma}[\rho] \leftarrow \Pi x : L_{N,\sigma}[\rho], L_{N,\sigma}[\rho] \cdot L_{N,\sigma}[\rho]\) is derivable. Note that the unlock constructor takes an atomic term as its main argument, thus avoiding the creation of possible \(\Sigma\)-redexes under substitution. Moreover, since unlocks can only receive locked terms in their body, no abstractions can ever arise. In Definition 2.3, we formalize the notion of \(\eta\)-expansion of a judgement, together with correspondence theorems between LLF_\varphi and CLLF_\varphi.

We present CLLF_\varphi in a fully-typed style, i.e. à la Church, but we could also follow [13] and present a version à la Curry, where the canonical forms \(\lambda x. M\) and \(L_M[N]\) do not carry type information. The type rules would then be, e.g.:

\[\Gamma ; x : \sigma \vdash \lambda x. M \Leftarrow \tau \quad (O-Abs)\]

\[\Gamma \vdash \Sigma \lambda x. M \Leftarrow \Pi x : \sigma. \tau \quad (O-Lock)\]

This latter syntax is more suitable in implementations because it optimises the notation. Following [17], we stick to the typeful syntax because it allows for a more direct comparison with non-canonical systems. This, however, is technically immaterial. Since judgements in canonical systems have unique derivations, one can show by induction on derivations that any provable judgement in the system where object terms are à la Curry has a unique type decoration of its object subterms, which turns it into a provable judgement in the version à la Church. Vice versa, any provable judgement in the version à la Church can forget the types in its object subterms, yielding a provable judgement in the version à la Curry.
Substitution in Atomic Objects
\[ \frac{c[M_0/x_0]_{\rho_0} = c}{(\mathcal{F} \cdot \text{O-Const})} \quad \frac{x_0[M_0/x_0]_{\rho_0} = M_0 : \rho_0}{(\mathcal{F} \cdot \text{O-Var-H})} \quad \frac{x \neq x_0}{(\mathcal{F} \cdot \text{O-Var})} \]
\[ A_1[M_0/x_0]_{\rho_0} = \lambda x_2 M'_1 : \rho_2 \quad M_2[M_0/x_0]_{\rho_0} = M'_2 \quad M_1[M_2/x_2]_{\rho_2} = M' \]  
\[ (A_1M_2)[M_0/x_0]_{\rho_0} = M' : \rho \]  
\[ (\mathcal{F} \cdot \text{O-App-H}) \]
\[ A_1[M_0/x_0]_{\rho_0} = A'_1 \quad M_2[M_0/x_0]_{\rho_0} = M'_2 \]  
\[ (A_1M_2)[M_0/x_0]_{\rho_0} = A'_1M'_2 \]  
\[ (\mathcal{F} \cdot \text{O-App}) \]

Substitution in Canonical Objects
\[ \frac{A[M_0/x_0]_{\rho_0} = A'}{(\mathcal{F} \cdot \text{O-R})} \quad \frac{A[M_0/x_0]_{\rho_0} = M' : \rho}{(\mathcal{F} \cdot \text{O-R-H})} \quad \frac{M[M_0/x_0]_{\rho_0} = M'}{\lambda x : \sigma[M][M_0/x_0]_{\rho_0} = \lambda x : \sigma[M']}{(\mathcal{F} \cdot \text{O-Abs})} \]
\[ \sigma_1[M_0/x_0]_{\rho_0} = \sigma'_1 \quad M_1[M_0/x_0]_{\rho_0} = M'_1 \quad M_2[M_0/x_0]_{\rho_0} = M'_2 \]  
\[ \mathcal{L}_{\mathcal{M}, \sigma_1}[M_2][M_0/x_0]_{\rho_0} = \mathcal{L}_{\mathcal{M}, \sigma_1}[M_2] \]  
\[ (\mathcal{F} \cdot \text{O-Lock}) \]

Substitution in Contexts
\[ \frac{[M_0/x_0]_{\rho_0}^C}{(\mathcal{F} \cdot \text{Ctx-Empty})} \quad \frac{x_0 \neq x}{(\mathcal{F} \cdot \text{Ctx-Term})} \]

Figure 5: Hereditary substitution, objects and contexts of CLLF \( \mathcal{P} \)

2.2 The Metatheory of CLLF \( \mathcal{P} \)

We start by studying the basic properties of hereditary substitution and the type system. First of all, we need to assume that the predicates are well-behaved in the sense of [18]. In the context of canonical systems, this notion needs to be rephrased as follows:

**Definition 2.1** (Well-behaved predicates for canonical systems). A finite set of predicates \( \{ \mathcal{P}_i \}_{i \in I} \) is well-behaved if each \( \mathcal{P}_i \) in the set satisfies the following conditions:

1. **Closure under signature and context weakening and permutation:**
   (a) If \( \Sigma \) and \( \Omega \) are valid signatures such that \( \Sigma \subseteq \Omega \) and \( \mathcal{P}(\Gamma \vdash \Sigma N \equiv \sigma) \), then \( \mathcal{P}(\Gamma \vdash \Omega N \equiv \sigma) \).
   (b) If \( \Gamma \) and \( \Delta \) are valid contexts such that \( \Gamma \subseteq \Delta \) and \( \mathcal{P}(\Gamma \vdash \Sigma \Delta N \equiv \sigma) \), then \( \mathcal{P}(\Delta \vdash \Sigma \Delta N \equiv \sigma) \).

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

As canonical systems do not feature reduction, the “classical” third constraint for well-behaved predicates (closure under reduction) is not needed here. Moreover, the second condition (closure under substitution) becomes “closure under hereditary substitution”.

**Lemma 2.1** (Decidability of hereditary substitution).

1. For any \( T \) in \( \{ \mathcal{F}, \mathcal{F}', \mathcal{F}, \mathcal{F}' \} \), and any \( M, x, \rho, \alpha \), it is decidable whether there exists a \( T' \) such that \( T[M/x]_{\rho}^\alpha = T' \) or there is no such \( T' \).
2. For any \( M, x, \rho, \alpha \), and \( A \), it is decidable whether there exists an \( A' \), such that \( A[M/x]_{\rho}^\alpha = A' \), or there exist \( M' \) and \( \rho' \), such that \( A[M/x]_{\rho}^\alpha = M' : \rho' \), or there are no such \( A' \) and \( M' \).
**Lemma 2.2** (Head substitution size). If \( A[M_0/x_0]_{\rho_0}^\alpha = M : \rho \), then \( \rho \) is a subexpression of \( \rho_0 \).

**Proof.** By induction on the derivation of \( A[M_0/x_0]_{\rho_0}^\alpha = M : \rho \). \( \square \)

**Lemma 2.3** (Uniqueness of substitution and synthesis).

1. It is not possible that \( A[M_0/x_0]_{\rho_0}^\alpha = A' \) and \( A[M_0/x_0]_{\rho_0}^\alpha = M : \rho \).
2. For any \( T \), if \( T[M_0/x_0]_{\rho_0}^m = T' \), and \( T[M_0/x_0]_{\rho_0}^m = T'' \), 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.** From the definition of hereditary substitution and the CLLF \( \rho \) type system. \( \square \)

**Lemma 2.4** (Composition of hereditary substitution). Let \( x \not\in x_0 \) and \( x \not\in \text{Fv}(M_0) \). Then:

1. For all \( T' \in \{ \text{\#}, \text{\&}, \text{\&}_a, \text{\&}_b, \text{\&} \} \), if \( M_2[M_0/x_0]_{\rho_0}^\alpha = M'_2 \), \( T_1[M_2/x]_{\rho_2}^m = T'_1 \), and \( T_1[M_0/x_0]_{\rho_0}^m = T''_1 \), then there exists a \( T : T[T_1[M_0/x_0]_{\rho_0}^m = T, \text{and } T'[M_2/x]_{\rho_2}^m = T' \).
2. If \( M_2[M_0/x_0]_{\rho_0}^\alpha = M'_2 \), \( A_1[M_2/x]_{\rho_2}^\alpha = M : \rho \), and \( A_1[M_0/x_0]_{\rho_0}^\alpha = A \), then there exists an \( M' : M[M_0/x_0]_{\rho_0}^\alpha = M' \), and \( A[M_2/x]_{\rho_2}^\alpha = M' : \rho \).
3. If \( M_2[M_0/x_0]_{\rho_0}^\alpha = M'_2 \), \( A_1[M_2/x]_{\rho_2}^\alpha = A \), and \( A_1[M_0/x_0]_{\rho_0}^\alpha = M : \rho \), then there exists an \( M' : A[M_0/x_0]_{\rho_0}^\alpha = M' : \rho \), and \( A[M_2/x]_{\rho_2}^\alpha = M' \).

By induction on derivations, similar to one in [13] p.14–15, we can prove:

**Theorem 2.5** (Transitivity). Let \( \Sigma \) \( \vdash \Sigma \Gamma, x_0 : \rho_0, \Gamma' \) and \( \Gamma \vdash \Sigma M_0 \Leftrightarrow \rho_0 \), and assume that all predicates are well-behaved. Then,

1. There exists a \( \Gamma'' : [M_0/x_0]_{\rho_0}^C = \Gamma'' \) and \( \Gamma \vdash \Sigma \Gamma, \Gamma'' \).
2. If \( \Gamma, x_0 : \rho_0, \Gamma' \vdash \Sigma K \) then there exists a \( K' : [M_0/x_0]_{\rho_0}^K = K' \) and \( \Gamma, \Gamma' \vdash \Sigma K' \).
3. If \( \Gamma, x_0 : \rho_0, \Gamma' \vdash \Sigma \sigma \) type, then there exists a \( \sigma' : [M_0/x_0]_{\rho_0}^F \sigma = \sigma' \) and \( \Gamma, \Gamma' \vdash \Sigma \sigma' \) type.
4. If \( \Gamma, x_0 : \rho_0, \Gamma' \vdash \Sigma M \Leftrightarrow \sigma \), then there exists \( \sigma' : [M_0/x_0]_{\rho_0}^F \sigma = \sigma' \) and \( [M_0/x_0]_{\rho_0}^\alpha M = M' \) and \( \Gamma, \Gamma' \vdash \Sigma M' \Leftrightarrow \sigma' \).

**Theorem 2.6** (Decidability of typing). If predicates in CCLL \( \rho \) are decidable, then all of the judgements of the system are decidable.

**Proof.** By induction on the complexity of judgements. \( \square \)

We can now precisely state the relationship between CCLL \( \rho \) and the LLF \( \rho \) system of [19]:

**Theorem 2.7** (Soundness). For any predicate \( \mathcal{P} \) of CCLL \( \rho \), we define a corresponding predicate in LLF \( \rho \) as follows: \( \mathcal{P}(\Gamma \vdash \Sigma M : \sigma) \) holds if and only if \( \Gamma \vdash \Sigma M : \sigma \) is derivable in LLF \( \rho \) and \( \mathcal{P}(\Gamma \vdash \Sigma M \Leftrightarrow \sigma) \) holds in CCLL \( \rho \). Then, we have:

1. If \( \Sigma \) sig is derivable in CCLL \( \rho \), then \( \Sigma \) sig is derivable in LLF \( \rho \).
2. If \( \vdash \Sigma \Gamma \) is derivable in CCLL \( \rho \), then \( \vdash \Sigma \Gamma \) is derivable in LLF \( \rho \).
3. If \( \vdash \Sigma \Gamma \) is derivable in CCLL \( \rho \), then \( \vdash \Sigma \Gamma \) is derivable in LLF \( \rho \).
4. If \( \vdash \Sigma \alpha \Rightarrow K \) is derivable in CCLL \( \rho \), then \( \vdash \Sigma \alpha : K \) is derivable in LLF \( \rho \).
5. If \( \vdash \Sigma \sigma \) type is derivable in CCLL \( \rho \), then \( \vdash \Sigma \sigma : \) type is derivable in LLF \( \rho \).
6. If \( \vdash \Sigma \alpha \Rightarrow \sigma \) is derivable in CCLL \( \rho \), then \( \vdash \Sigma A : \sigma \) is derivable in LLF \( \rho \).
7. If \( \vdash \Sigma M \Leftrightarrow \sigma \) is derivable in CCLL \( \rho \), then \( \vdash \Sigma M : \sigma \) is derivable in LLF \( \rho \).
Vice versa, all LLF\(\varphi\) judgements in long \(\beta\eta\)-normal form (\(\beta\eta\)-lnf) are derivable in CLLF\(\varphi\). The definition of a judgement in \(\beta\eta\)-lnf is based on the following extension of the standard \(\eta\)-rule to the lock constructor \(\lambda x:\sigma. Mx \to_{\eta} M\) and \(\mathcal{L}_{N,\sigma}^{\varphi}[\mathcal{U}_{N,\sigma}[M]] \to_{\eta} M\).

**Definition 2.2.** An occurrence \(\xi\) of a constant or a variable in a term of an LLF\(\varphi\) judgement is fully applied and unlocked w.r.t. its type or kind \(\Pi \bar{x}_1: \bar{\sigma}_1, \ldots, \Pi \bar{x}_n: \bar{\sigma}_n, \bar{\xi}\)[\(\alpha\)]…, where \(\bar{\sigma}_1, \ldots, \bar{\sigma}_n\) are vectors of locks, if \(\xi\) appears only in contexts that are of the form \(\mathcal{U}_n((\ldots(\mathcal{U}_1[\xi M_1])\ldots)M_n)\), where \(\bar{M}_1, \ldots, \bar{M}_n, \bar{\mathcal{U}}_1, \ldots, \bar{\mathcal{U}}_n\) have the same arities of the corresponding vectors of \(\Pi\)'s and locks.

**Definition 2.3** (Judgements in long \(\beta\eta\)-normal form).

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

**Theorem 2.8** (Correspondence). Assume that all predicates in LLF\(\varphi\) are well-behaved, according to Definition 2.1 [18]. For any predicate \(\mathcal{P}\) in LLF\(\varphi\), we define a corresponding predicate in CLLF\(\varphi\) with: \(\mathcal{P}(\Gamma \vdash_{\Sigma} M \leftarrow \sigma)\) holds if \(\Gamma \vdash_{\Sigma} M \leftarrow \sigma\) is derivable in CLLF\(\varphi\) and \(\mathcal{P}(\Gamma \vdash_{\Sigma} M : \sigma)\) holds in LLF\(\varphi\).

Then, we have:

1. If \(\Sigma\) sig is in \(\beta\eta\)-Inf and is LLF\(\varphi\)-derivable, then \(\Sigma\) sig is CLLF\(\varphi\)-derivable.
2. If \(\vdash_{\Sigma} \Gamma\) is in \(\beta\eta\)-Inf and is LLF\(\varphi\)-derivable, then \(\vdash_{\Sigma} \Gamma\) is CLLF\(\varphi\)-derivable.
3. If \(\vdash_{\Sigma} K\) is in \(\beta\eta\)-Inf, and is LLF\(\varphi\)-derivable, then \(\vdash_{\Sigma} K\) is CLLF\(\varphi\)-derivable.
4. If \(\vdash_{\Sigma} \alpha : K\) is in \(\beta\eta\)-Inf, and is LLF\(\varphi\)-derivable, then \(\vdash_{\Sigma} \alpha \Rightarrow K\) is CLLF\(\varphi\)-derivable.
5. If \(\vdash_{\Sigma} \sigma:\text{type}\) is in \(\beta\eta\)-Inf, and is LLF\(\varphi\)-derivable, then \(\vdash_{\Sigma} \sigma\) type is CLLF\(\varphi\)-derivable.
6. If \(\vdash_{\Sigma} A : \alpha\) is in \(\beta\eta\)-Inf, and is LLF\(\varphi\)-derivable, then \(\vdash_{\Sigma} A \Rightarrow \alpha\) is CLLF\(\varphi\)-derivable.
7. If \(\vdash_{\Sigma} M : \sigma\) is in \(\beta\eta\)-Inf, and is LLF\(\varphi\)-derivable, then \(\vdash_{\Sigma} M \leftarrow \sigma\) is CLLF\(\varphi\)-derivable.

**Proof.** Follows closely the proof of the corresponding Correspondence Theorem 7 [17].

Notice that, by the Correspondence Theorem above, any well-behaved predicate \(\mathcal{P}\) in LLF\(\varphi\) in the sense of Definition 2.1 [18] induces a well-behaved predicate in CLLF\(\varphi\). Finally, notice that not all LLF\(\varphi\) judgements have a corresponding \(\beta\eta\)-Inf. Namely, the judgement \(x: \mathcal{L}_{N,\sigma}^{\varphi}[\rho] \vdash_{\Sigma} x: \mathcal{L}_{N,\sigma}^{\varphi}[\rho]\) does not admit an \(\eta\)-expanded normal form when the predicate \(\mathcal{P}\) does not hold on \(\bar{N}\), as the rule \((O\text{-Unlock})\) can be applied only when the predicate holds.

### 3 The Type System CLLF\(\varphi\)?

The main idea behind CLLF\(\varphi\) (see Figures 6, 7, and 8)\footnote{For lack of space, we present in these figures only the categories and rules of CLLF\(\varphi\) that differ from their CLLF\(\varphi\) counterparts.} is to “empower” the framework of CLLF\(\varphi\) by adding to the lock/unlock mechanism the possibility to receive from the external oracle a witness satisfying suitable constraints. Thus, we can pave way for the gluing together of different proof development environments beyond proof irrelevance scenarios. In this context, the lock constructor behaves as a binder. The new \((O\text{-Lock})\) rule is the following:

\[
\frac{\Gamma, x: \sigma \vdash_{\Sigma} M \leftarrow \rho}{\vdash_{\Sigma} \mathcal{L}_{\Sigma,\sigma}^{\varphi}[M] \leftarrow \mathcal{L}_{\Sigma,\sigma}^{\varphi}[\rho]}
\]
The term $\U_{\sigma}$ of a given type $M$. Indeed, that of recording the need to delegate to the external oracle the inference of a suitable witness of a given
dunction theorem since we did not introduce a non-canonical variant
The Metatheory of CLLF will be "released" in
In this section, we discuss the encodings of a collection of logical systems which illustrate the expressive
4 Case studies
where the variable $x$ is a placeholder bound in $M$ and $\rho$, which will be replaced by the concrete term that will be returned by the external oracle call. The intuitive meaning behind the (O-Lock) rule is, therefore, that of recording the need to delegate to the external oracle the inference of a suitable witness of a given type. Indeed, $M$ can be thought of as an "incomplete" term which needs to be completed by an inhabitant of a given type $\sigma$ satisfying the constraint $\mathcal{P}$. The actual term, possibly synthesized by the external tool, will be "released" in CLLF, by the unlock constructor in the (O-Lock) rule as follows:
\[
\Gamma \vdash_{\Sigma} \U_{\sigma} \Rightarrow L_{\lambda \sigma} \rho \quad \Rightarrow \quad \Gamma \vdash_{\Sigma} N \iff \sigma \quad \mathcal{P}(\Gamma \vdash_{\Sigma} N \iff \sigma)
\]
The term $\U_{\sigma}$ intuitively means that $N$ is precisely the synthesized term satisfying the constraint $\mathcal{P}(\Gamma \vdash_{\Sigma} N \iff \sigma)$ that will replace in CLLF all the free occurrences of $x$ in $\rho$. This replacement is executed in the (SO-Lock-H) hereditary substitution rule (Figure 6).

Similarly to CLLF, in CLLF it is also possible to "postpone" or delay the verification of an external predicate. Whence, the synthesis of the actual inhabitant $N$ can be delayed, thanks to the (O-Nested-Unlock) rule:
\[
\Gamma, y : \tau \vdash_{\Sigma} L_{\lambda \sigma} \rho \quad \Rightarrow \quad \Gamma \vdash_{\Sigma} A \Rightarrow L_{\lambda \sigma} \tau \quad \rho[\U_{\sigma}[A]/y]_{(\tau)}^{F} = \rho' \quad M[\U_{\sigma}[A]/y]_{(\tau)}^{O} = M'
\]
The Metatheory of CLLF follows closely that of CLLF as far as decidability. We have no correspondence theorem since we did not introduce a non-canonical variant CLLF. This could have been done similarly to LLF.

4 Case studies
In this section, we discuss the encodings of a collection of logical systems which illustrate the expressive power and the flexibility of CLLF and CLLF. We discuss Fitch-Prawitz Consistent Set theory, FPST
Substitution in Canonical Families
\[
\sigma_1[M_0/x_0]^{F}_{\rho_0} = \sigma'_1, \quad \sigma_2[M_0/x_0]^{F}_{\rho_0} = \sigma'_2 \quad \Rightarrow \quad \mathcal{L}^{\mathcal{P}}_{x,\sigma_2}[\sigma_2][M_0/x_0]^{F}_{\rho_0} = \mathcal{L}^{\mathcal{P}}_{x,\sigma'_2}[\sigma'_2]
\]  
(\mathcal{P}-F-Lock)

Substitution in Atomic Objects
\[
\sigma[M_0/x_0]^{F}_{\rho_0} = \sigma' \quad M[M_0/x_0]^{O}_{\rho_0} = M' \quad M_1[M'/x]^{\sigma'}_{\rho_0} = M_2 \quad A[M_0/x_0]^{o}_{\rho_0} = \mathcal{L}^{\mathcal{P}}_{x,\sigma'}[M_1] : \mathcal{L}^{\mathcal{P}}_{x,\sigma'}[\rho]
\]  
(\mathcal{F}-O-Unlock-H)

Substitution in Canonical Objects
\[
\sigma_1[M_0/x_0]^{F}_{\rho_0} = \sigma'_1, \quad M_1[M_0/x_0]^{O}_{\rho_0} = M'_1 \quad \Rightarrow \quad \mathcal{L}^{\mathcal{P}}_{x,\sigma_1}[M_1][M_0/x_0]^{O}_{\rho_0} = \mathcal{L}^{\mathcal{P}}_{x,\sigma'_1}[M'_1]
\]  
(\mathcal{F}-O-Lock)

![Image: CLLF_p & CLLF_p? Hereditary Substitution — changes w.r.t. CLLF_p]

[29], some applications of FPST to normalizing \(\lambda\)-calculus, a system of Light Linear Logic in CLLF_p, and an the encoding of a partial function in CLLF_p?.

The crucial step in encoding a logical system in CLLF_p or CLLF_p? is to define the predicates involved in locks. Predicates defined on closed terms are usually unproblematic. Difficulties arise in enforcing the properties of closure under hereditary substitution and closure under signature and context extension, when predicates are defined on open terms. To be able to streamline the definition of well-behaved predicates we introduce the following:

**Definition 4.1.** Given a signature \(\Sigma\) let \(\Lambda_\Sigma\) (respectively \(\Lambda_\Sigma^c\)) be the set of LLF_p terms (respectively closed LLF_p terms) definable using constants from \(\Sigma\). A term \(M\) has a skeleton in \(\Lambda_\Sigma\) if there exists a term \(N[x_1,\ldots,x_n] \in \Lambda_\Sigma\), whose free variables (called holes of the skeleton) are in \(\{x_1,\ldots,x_n\}\), and there exist terms \(M_1,\ldots,M_n\) such that \(M \equiv N[M_1/x_1,\ldots,M_n/x_n]\).

### 4.1 Fitch Set Theory à la Prawitz - FPST

In this section, we present the encoding of a formal system of remarkable logical as well as historical significance, namely the system of consistent Naïve Set Theory, FPST, introduced by Fitch [10]. This system was first presented in Natural Deduction style by Prawitz [29]. As Naïve Set Theory is inconsistent, to prevent the derivation of inconsistencies from the unrestricted abstraction rule, only normalizable deductions are allowed in FPST. Of course, this side-condition is extremely difficult to capture using traditional tools.

In the present context, instead, we can put to use the machinery of CLLF_p to provide an appropriate encoding of FPST where the global normalization constraint is enforced locally by checking the proof-object. This system is a beautiful illustrative example of the bag of tricks that CLLF_p supports. Checking that a proof term is normalizable would be the obvious predicate to use in the corresponding lock-type, but this would not be a well-behaved predicate if free variables, i.e. assumptions, are not sterilized. To this end, we introduce a distinction between generic judgements, which cannot be directly utilized in arguments, but which can be assumed, and apodictic judgements, which are directly involved in proof rules. In order to make use of generic judgements, one has to downgrade them to an apodictic one. This is achieved by a suitable coercion function.

**Definition 4.2** (Fitch Prawitz Set Theory, FPST). The full system of Fitch (see [29]), as presented by Prawitz is defined into Appendix [A] here we only give the crucial rules for implication and for set-
\[ \Gamma, A \vdash_{\text{FPST}} B (\square I) \]

In Fitch’s system, FPST, conjunction and universal quantification are defined as usual, while negation is defined constructively, but it still allows for the usual definitions of disjunction and existential quantification. What makes FPST consistent is that not all standard deductions in FPST are legal. Standard deductions are called quasi-deductions in FPST. A legal deduction in FPST is defined instead, as a quasi-deduction which is normalizable in the standard sense of Natural Deduction, namely it can be transformed in a derivation where all elimination rules occur before introductions.

**Definition 4.3** (LLF $\varphi$ signature $\Sigma_{\text{FPST}}$ for Fitch Prawitz Set Theory). The following constants are introduced:

- \( \odot : \text{Type} \)
- \( \theta : \text{Type} \)
- \( \Pi : \text{Type} \)
- \( \Lambda : \Pi (\text{Type}) \)
- \( \lambda : (\iota \rightarrow \odot) \rightarrow \theta \)
- \( \epsilon : (\iota \rightarrow \iota) \rightarrow \odot \)
- \( \square : (\theta \rightarrow \odot) \rightarrow (\odot \rightarrow \theta) \)
- \( \sigma : \text{Type} \)
- \( \tau : \text{Type} \)
- \( \rho : \text{Type} \)

where \( \odot \) is the type of propositions, \( \square \) and the “membership” predicate \( \epsilon \) are the syntactic constructors for propositions, \( \lambda \) is the “abstraction” operator for building “sets”, \( \Pi \) is the apodictic judgement, \( \Lambda \) is the coercer function, and \( \langle x, y \rangle \) denotes the encoding of pairs, whose type is denoted by \( \sigma \times \tau \), e.g. \( \lambda u: \sigma \rightarrow \tau \rightarrow \rho. u \times y : (\sigma \rightarrow \tau \rightarrow \rho) \rightarrow \rho \). The predicate in the lock is defined as follows:

\[ \boxed{\text{Fitch}(\Gamma \vdash_{\Sigma_{\text{FPST}}} \langle x, y \rangle \iff T(\odot \times T(\odot \times \theta)))} \]

it holds iff \( x \) and \( y \) have skeletons in \( \Lambda_{\Sigma_{\text{FPST}}} \), all the holes of which have either type \( \odot \) or are guarded by a \( \delta \), and hence have type \( \Pi(\odot) \), and, moreover, the proof derived by combining the skeletons of \( x \) and \( y \) is normalizable in the natural sense.

For lack of space, we do not spell out the rules concerning the other logical operators, because they are all straightforward provided we use only the apodictic judgement \( T(\cdot) \), but a few remarks are mandatory. The notion of normalizable proof is the standard notion used in natural deduction. The predicate Fitch is well-behaved because it considers terms only up-to holes in the skeleton, which can have type \( \odot \) or are generic judgements. Adequacy for this signature can be achieved in the format of [18]:

**Theorem 4.1** (Adequacy for Fitch-Prawitz Naive Set Theory). If \( A_1, \ldots, A_n \) are the atomic formulas occurring in \( B_1, \ldots, B_m, A \), then \( B_1 \ldots B_m \vdash_{\text{FPST}} A \) iff there exists a normalizable \( M \) such that \( A_1: \odot, \ldots, A_n: \odot, x_1: V(B_1), \ldots, x_m : V(B_m) \vdash_{\Sigma_{\text{FPST}}} M \iff T(A) \) (where \( A \) and \( B_1 \) represent the encodings of, respectively, \( A \) and \( B_1 \) in CLLF $\varphi$, for \( 1 \leq i \leq m \)).

### 4.2 A Type System for strongly normalizing $\lambda$-terms

Fitch-Prawitz Set Theory, FPST, is a rather intriguing, albeit unexplored, set theoretic system. The normalizability criterion for accepting a quasi-deduction prevents the derivation of contradictions, and
Let equality be defined as Leibniz equality, then, assuming \( \Lambda \):

Then, there exists a formula \( B \) of Theorem 4.2 (Fixpoint) originally given by J-Y. Girard for Light Linear Logic, as follows:

The strongly normalizing \( \lambda \)-expressive type system which encompasses all kinds of quantification.

In this subsection, we sketch how to use FPST to define a type system which can type precisely all the strongly normalizing \( \lambda \)-terms. First we need to adapt to FPST, Proposition 4, Appendix A.1 of [12], originally given by J-Y. Girard for Light Linear Logic, as follows:

**Theorem 4.2** (Fixpoint). Let \( A[P,x_1,\ldots,x_n] \) be a formula of FPST with an \( n \)-ary predicate variable \( P \). Then, there exists a formula \( B \) of FPST, such that there exists a normalizable deduction in FPST between \( A[\lambda x_1,\ldots,x_n,B[x_1,\ldots,x_n],x_1,\ldots,x_n] \) and \( B \), and viceversa.

**Proof.** Let equality be defined as Leibniz equality, then, assuming \( n = 1 \), define \( \Lambda \equiv \lambda z.\exists x.\exists y.z = \langle x,y \rangle \& A[(\lambda w.\langle w,y \rangle \in y),x] \). Then \( \langle x,\Lambda \rangle \in \Lambda \) is equivalent, in the sense of FPST, to \( A[(\lambda w.\langle w,\Lambda \rangle \in \Lambda),x] \).

The above Fixpoint Theorem can be used to define recursive functions and inductive definitions. Hence, let us consider a concrete representation, possibly over natural numbers, of the terms of \( \lambda \)-calculus in FPST, and call it \( \Lambda_0 \). It exists, because even if the theory is non-standard, it allows the definition of some putative notion of a natural number. Using again the Fixed Point Theorem, we can define a fixpoint \( \Lambda \) such that \( x \in \Lambda \) is equivalent, in the sense of Fitch, to \( x \in \Lambda_0 \& \forall y.y \in \Lambda_0 \subset app(x,y) \in \Lambda \). Here, \( app(x,y) \) denotes the concrete representation of “applying” \( x \) to \( y \). It is now straightforward to check that only normalizing terms can “be typed by \( \Lambda \)”, i.e. belong to \( \Lambda \). Since one can derive in FPST that a representation of a \( \lambda \)-term, say \( M \), belongs to \( \Lambda \), only if there is a normalizable derivation of \( M \in \Lambda \), then there is a natural reflection of the normalizability of the FPST derivation of the typing judgement \( M \in \Lambda \), and the fact that the term represented by \( M \) is indeed normalizable.

### 4.3 A Normalizing call-by-value \( \lambda \)-calculus

In this section we sketch how to express in CLLF \( \rightarrow \) a call-by-value \( \lambda \)-calculus where \( \beta \)-reductions fire only if the result is normalizing.

**Definition 4.4** (Normalizing call-by-value \( \lambda \)-calculus, \( \Sigma_{\lambda,N} \)).

\[
\begin{align*}
o & : \text{Type} & \text{Eq} & : o \to o \to \text{Type} & \text{app} & : o \to o \to o \\
v & : \text{Type} & \text{var} & : v \to o & \text{lam} & : (v \to o) \to o \\
c_{\beta} \in \Pi M:o \to o \vdash N:o.\mathcal{Z}^{c_{\beta}}_{\Sigma_{\lambda,N}}[M,N,\{(o \to o) \times o\}][\text{Eq} \ (\text{app} \ (\text{lam} \ \lambda x:v.M(\text{var} \ x)) \ N) \ (M N)]
\end{align*}
\]

where the predicate \( \mathcal{Z}^{c_{\beta}} \) holds on \( \Gamma \vdash_{\Sigma_{\lambda,N}} (M,N) \leftarrow (o \to o) \times o \) if both \( M \) and \( N \) have skeletons in \( \Lambda_{\Sigma_{\lambda,N}} \) whose holes are guarded by a \( \text{var} \) and, moreover, \( MN \) “normalizes”, in the intuitive sense, outside terms guarded by a \( \text{var} \).

### 4.4 Elementary Affine Logic

In this section we give a shallow encoding of Elementary Affine Logic as presented in [2]. This example will exemplify how locks can be used to deal with global syntactic constraints as in the promotion rule of Elementary Affine Logic.

**Definition 4.5** (Elementary Affine Logic [2]). Elementary Affine Logic can be specified by the following rules:
\[
\begin{align*}
A \vdash_{\text{EAL}} A & \quad \text{(Var)} \\
\Gamma, A \vdash_{\text{EAL}} B & \quad \text{(Weak)} \\
\Gamma, \Delta \vdash_{\text{EAL}} A \rightarrow B & \quad \text{(Contr)} \\
\Gamma, \Delta \vdash_{\text{EAL}} B & \quad \text{(App)} \\
\Gamma \vdash_{\text{EAL}} ! A & \quad \text{(Abst)} \\
\Gamma \vdash_{\text{EAL}} ! A_1 \ldots ! A_n & \quad \text{(PROM)}
\end{align*}
\]

**Definition 4.6** (LLF_\rho signature \(\Sigma_{EAL}\) for Elementary Affine Logic).

\[o : \text{Type} \quad T : o \rightarrow \text{Type} \quad V : o \rightarrow \text{Type} \quad \rightarrow : o \rightarrow o \rightarrow o \quad ! : o \rightarrow o\]

\[c_{\text{appl}} : \Pi A, B : o. \ T(A) \rightarrow (T(A \rightarrow B) \rightarrow T(B)) \quad c_{\text{val}} : \Pi A : o. \ V(A) \rightarrow (T(!A))\]

\[c_{\text{abstr}} : \Pi A, B : o. \ \Pi x : (T(A) \rightarrow T(B)) \rightarrow (\mathcal{L}^{\text{Closed}}_{x, T(A) \rightarrow T(B)}[T(A \rightarrow B)])\]

\[c_{\text{promV}} : \Pi A, B, C : o. \ \Pi x : (V(A \rightarrow B) \rightarrow \mathcal{L}^{\text{Closed}}_{x, V(A \rightarrow B)}[T(A \rightarrow V(B))]\]

where \(o\) is the type of propositions, \(\rightarrow\) and \(!\) are the obvious syntactic constructors, \(T\) is the basic judgement, and \(V(\cdot)\) is an auxiliary judgement. The predicates involved in the locks are defined as follows:

- **Light** (\(\Gamma \vdash_{\Sigma_{EAL}} x \equiv T(A) \rightarrow T(B)\)) holds iff if \(A\) is not of the shape \(!A\) then the bound variable of \(x\) occurs at most once in the normal form of \(x\).
- **Closed** (\(\Gamma \vdash_{\Sigma_{EAL}} x \equiv T(A)\)) holds iff the skeleton of \(x\) contains only free variables of type \(o\), i.e., no variables of type \(T(B)\), for any \(B : o\).

A few remarks are mandatory. The promotion rule in [2] is in effect a family of natural deduction rules with a growing number of assumptions. Our encoding achieves this via the auxiliary judgement \(V(\cdot)\), the effect of which is self-explanatory. Adequacy for this signature can be achieved only in the format of [13], namely:

**Theorem 4.3** (Adequacy for Elementary Affine Logic). *If \(A_1, \ldots, A_n\) are the atomic formulas occurring in \(B_1, \ldots, B_m, A\), then \(B_1 \ldots B_m \vdash_{\text{EAL}} A\) iff there exists \(M\) and \(A_1 : o, \ldots, A_n : o, x_1 : T(B_1), \ldots, x_m : T(B_m) \vdash_{\Sigma_{EAL}} M \equiv T(A)\) (where \(A_1, B_1\) represent the encodings of, respectively, \(A\) and \(B_1\) in LLF_\rho, for \(1 \leq i \leq m\) and all variables \(x_1 \ldots x_m\) occurring more than once in \(M\) have type of the shape \(T(B_i) \equiv T(!C_i)\) for some suitable formula \(C_i\).*

The check on the context of the Adequacy Theorem is external to the system LLF_\rho, but this is in the nature of results which relate internal and external concepts. For example, the very concept of LLF_\rho context, which appears in any adequacy result, is external to LLF_\rho. Of course, this check is internalized if the term is closed.

### 4.5 Square roots of natural numbers in LLF_\rho

It is well-known that logical frameworks based on Constructive Type Theory do not permit definitions of non-terminating functions (i.e., all the functions one can encode in such frameworks are total). One interesting example of CLF_\rho system is the possibility of reasoning about partial functions by delegating their computation to external oracles, and getting back their possible outputs, via the lock-unlock mechanism of CLLF_\rho.

For instance, we can encode natural numbers and compute their square roots by means of the following signature (\((x, y)\) denotes the encoding of pairs, whose type is denoted by \(\sigma \times \tau\), and \(\text{fst}\) and \(\text{snd}\) are the first and second projections, respectively):

\[
\begin{align*}
nat : \text{type} & \quad 0 : \text{nat} \quad S : \text{nat} \rightarrow \text{nat} \quad \text{plus} : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat} \quad \text{minus} : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat} \\
mult : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat} \quad \text{sqroot} : \text{nat} \rightarrow \text{nat} \quad \text{eval} : \text{nat} \rightarrow \text{nat} \rightarrow \text{type} \\
\text{sqrt} : \Pi x : \text{nat}. \mathcal{L}_{\gamma, \text{nat} \times \sigma}[(\text{eval} (\text{sqroot} x) (\text{fst} y))]\end{align*}
\]
where \( \text{eval} \) represents the usual evaluation predicate, the variable \( y \) is a pair and
\[
\sigma \equiv (\text{eval} (\text{plus} (\text{minus} x (\text{mult} z z)) (\text{minus} (\text{mult} z z) x) 0))
\]
and \( \text{SQRT}(\Gamma \vdash y \leftarrow \text{nat} \times \sigma) \) holds if and only if the first projection of \( y \) is the minimum number \( N \) such that \( (x - N \times N) + (N \times N - x) = 0 \), where \( + \) and \( \times \) are represented by \text{plus} and \text{mult}, while \( - \) (represented by \text{minus} in our signature) is defined as follows:
\[
x - y \triangleq \begin{cases} 
  x - y & \text{if } x \geq y \\
  0 & \text{otherwise}
\end{cases}
\]
Thus, the specification of \text{sqrt} is not explicit in CLLF \&C LLF P?, since it is implicit in the definition of \text{SQRT}.

5 Related work

Building a universal framework with the aim of “gluing” different tools and formalisms together is a long standing goal that has been extensively explored in the inspiring work on Logical Frameworks by [4, 26, 32, 6, 25, 27, 28, 16]. Moreover, the appealing monadic structure and properties of the lock/unlock mechanism go back to Moggi’s notion of computational monads [24]. Indeed, our system can be seen as a generalization to a family of dependent lax operators of Moggi’s partial \( \lambda \)-calculus [23] and of the work carried out in [7, 22] (which is also the original source of the term “lax”). A correspondence between lax modalities and monads in functional programming was pointed out in [11]. On the other hand, although the connection between constraints and monads in logic programming was considered in the past, e.g., in [25, 9, 8], to our knowledge, our systems are the first attempt to establish a clear correspondence between side conditions and monads in a higher-order dependent-type theory and in logical frameworks. Of course, there are a lot of interesting points of contact with other systems in the literature which should be explored. For instance, in [25], the authors introduce a contextual modal logic, where the notion of context is rendered by means of monadic constructs. We only point out that, as we did in our system, they could have also simplified their system by doing away with the \text{let} construct in favor of a deeper substitution. Schröder-Heister has discussed in a number of papers, see e.g. [31, 30], various restrictions and side conditions on rules and on the nature of assumptions that one can add to logical systems to prevent the arising of paradoxes. There are some potential connections between his work and ours. It would be interesting to compare his requirements on side conditions being “closed under substitution” to our notion of well-behaved predicate. Similarly, there are commonalities between his distinction between specific and unspecific variables, and our treatment of free variables in well-behaved predicates.

References

### Fitch Set Theory

**Definition A.1** (Fitch Prawitz Set Theory). The full system of Fitch (see [29]), as presented by Prawitz is defined as follows:

\[
\frac{\Gamma, A \vdash_{FST} B}{\Gamma \vdash_{FST} A \supset B} \quad (\supset I) \\
\frac{\Gamma \vdash_{FST} A \quad \Gamma \vdash_{FST} A \supset B}{\Gamma \vdash_{FST} B} \quad (\supset E) \\
\frac{\Gamma \vdash_{FST} A T/\lambda x A}{\Gamma \vdash_{FST} T \in \lambda x A} \quad (\lambda I) \\
\frac{\Gamma \vdash_{FST} T \in \lambda x A}{\Gamma \vdash_{FST} A T/\lambda x A} \quad (\lambda E) \\
\frac{\Gamma \vdash_{FST} A \quad \Gamma \vdash_{FST} B}{\Gamma \vdash_{FST} A \wedge B} \quad (&I) \\
\frac{\Gamma \vdash_{FST} A \wedge B}{\Gamma \vdash_{FST} A} \quad (&E1) \\
\frac{\Gamma \vdash_{FST} A \wedge B}{\Gamma \vdash_{FST} B} \quad (&E2) \\
\frac{\Gamma \vdash_{FST} A}{\Gamma \vdash_{FST} A \vee B} \quad (\vee I) \\
\frac{\Gamma \vdash_{FST} B}{\Gamma \vdash_{FST} A \vee B} \quad (\vee I) \\
\frac{\Gamma \vdash_{FST} A \vee B \quad \Gamma \vdash_{FST} B}{\Gamma \vdash_{FST} C} \quad (\vee E) \\
\frac{\Gamma \vdash_{FST} A}{\Gamma \vdash_{FST} \forall x A} \quad (\forall I) \\
\frac{\Gamma \vdash_{FST} \forall x A}{\Gamma \vdash_{FST} A T/\lambda x A} \quad (\forall E) \\
\frac{\Gamma \vdash_{FST} A \quad \Gamma \vdash_{FST} A \lambda x A}{\Gamma \vdash_{FST} A} \quad (\forall I) \\
\frac{\Gamma \vdash_{FST} A \lambda x A}{\Gamma \vdash_{FST} A} \quad (\forall C) \\
\frac{\Gamma, \neg A \vdash_{FST} C}{\Gamma \vdash_{FST} \neg (A \wedge B)} \quad (\neg & I) \\
\frac{\Gamma \vdash_{FST} A \quad \Gamma, \neg B \vdash_{FST} C}{\Gamma \vdash_{FST} \neg (A \wedge B)} \quad (\neg & E) \\
\frac{\Gamma \vdash_{FST} \neg (A \supset B)}{\Gamma \vdash_{FST} \neg B} \quad (\neg \supset I) \\
\frac{\Gamma \vdash_{FST} A \quad \Gamma \vdash_{FST} \neg B}{\Gamma \vdash_{FST} \neg (A \supset B)} \quad (\neg \supset E) \\
\frac{\Gamma \vdash_{FST} \neg A}{\Gamma \vdash_{FST} \neg A} \quad (\neg E) \\
\frac{\Gamma \vdash_{FST} \neg A \quad \Gamma, \neg A T/\lambda x A}{\Gamma \vdash_{FST} \neg \exists x A} \quad (\neg \exists E) \\
\frac{\Gamma \vdash_{FST} \neg \forall x A \quad \Gamma, \neg A T/\lambda x A}{\Gamma \vdash_{FST} \neg \exists x A} \quad (\neg \exists E) \\
\frac{\Gamma \vdash_{FST} \neg A \quad \Gamma \vdash_{FST} \neg B}{\Gamma \vdash_{FST} \neg B} \quad (\neg \forall E) \\
\frac{\Gamma \vdash_{FST} \neg \exists x A \quad \Gamma \vdash_{FST} \neg \forall x A}{\Gamma \vdash_{FST} \neg \exists x A} \quad (\neg \forall E)
\]

As noticed in [29], negation in Fitch’s system is usually defined in a constructive way as follows:

\[
\frac{\Gamma \vdash_{FST} \neg A}{\Gamma \vdash_{FST} \neg (A \wedge B)} \quad (\neg & I1) \\
\frac{\Gamma \vdash_{FST} \neg (A \wedge B) \quad \Gamma \vdash_{FST} \neg B}{\Gamma \vdash_{FST} \neg (A \wedge B)} \quad (\neg & I2) \\
\frac{\Gamma \vdash_{FST} \neg (A \supset B)}{\Gamma \vdash_{FST} \neg B} \quad (\neg \supset I) \\
\frac{\Gamma \vdash_{FST} \neg (A \supset B) \quad \Gamma \vdash_{FST} \neg B}{\Gamma \vdash_{FST} \neg (A \supset B)} \quad (\neg \supset E) \\
\frac{\Gamma \vdash_{FST} \neg A}{\Gamma \vdash_{FST} \neg A} \quad (\neg E) \\
\frac{\Gamma \vdash_{FST} \neg A \quad \Gamma \vdash_{FST} \neg B}{\Gamma \vdash_{FST} \neg B} \quad (\neg \forall I) \\
\frac{\Gamma \vdash_{FST} \neg A \quad \Gamma \vdash_{FST} \neg B}{\Gamma \vdash_{FST} \neg B} \quad (\neg \forall E)
\]