Plugging-in Proof Development Environments using Locks in LF

Furio Honsell, Luigi Liquori, Petar Maksimovic, Ivan Scagnetto

Under consideration for publication in Math. Struct. in Comp. Science

Plugging-in Proof Development Environments using *Locks* in LF

Furio Honsell[1], Luigi Liquori[2], Petar Maksimović[3,4†] and Ivan Scagnetto[1]

[1] *Department of Mathematics and Computer Science University of Udine, Italy*
[2] *Inria Sophia Antipolis Méditerranée, France*
[3] *Imperial College London, UK*
[4] *Mathematical Institute of the Serbian Academy of Sciences and Arts, Serbia*

Received 11 February 2016

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 *plugging-in*, *i.e.* gluing together, different Type Theories and proof development environments. The oracle can be invoked either 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, $\mathsf{CLLF}_{\mathcal{P}}$, is the canonical version of the system $\mathsf{LLF}_{\mathcal{P}}$, presented earlier by the authors. The second system, $\mathsf{CLLF}_{\mathcal{P}^?}$, features the possibility of invoking the oracle to obtain also a witness satisfying a given constraint. We discuss encodings of Fitch-Prawitz Set theory, call-by-value λ-calculi, systems of Light Linear Logic, and partial functions.

1. Introduction

This work is an extended version of (Honsell *et al.* 2015) and is part of an ongoing research programme, (Honsell *et al.* 2012; Honsell 2013; Honsell *et al.* 2014; Honsell *et al.* 2016), aiming to define a simple *Universal Meta-language*, in the form of a an extension of the Constructive Type Theory LF, that can support the effects of *plugging-in*, *i.e.* connecting different proof development environments.

The basic idea underpinning these logical frameworks is to allow for the user to express explicitly the *invocation* of external tools and to uniformly *record* their *effects* by means of a new *monadic* type-constructor $\mathcal{L}^{\mathcal{P}}_{M,\sigma}[\cdot]$, 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, beforehand, to *verify* a constraint, which is written in the form of a *meta-predicate* on a judgement *i.e.* $\mathcal{P}(\Gamma \vdash_{\Sigma} M : \sigma)$. There are no limitations on how *external proof search tools* that have been plugged-in can supply such evidence. These can even make use of *oracles* or exploit some other epistemic source, such as diagrams, physical analogies,

† The work presented in this paper was partially supported by the Serbian Ministry of Education, Science, and Technological Development, projects ON174026 and III44006.

or explicit computations according to the *Poincaré Principle* (Barendregt *et al.* 2002; Kerber 2006). We can say, therefore, that locks subsume different *proof attitudes*, such as "proof-irrelevant" approaches, where one is interested only in knowing that some evidence does exist, as well as approaches relying on powerful terminating metalanguages. Indeed, locks allow for a straightforward accommodation of many different *proof cultures* within a single Logical Framework, which can otherwise be embedded only very deeply (Boulton *et al.* 1992; Hirschkoff 1997) or axiomatically (Honsell *et al.* 2001).

Pragmatically, using lock constructors, one can *factorize* the goal, produce pieces of evidence using different proof environments, and finally *glue* them back together, using the *unlock operator*, which *releases* the locked term in the calling framework. Clearly, the task of checking the validity of external evidence relies entirely on the external tool which has been plugged-in. Our framework limits itself to recording in the proof term that there has been a recourse to an external tool, by means of an *unlock* destructor.

In a departure from our earlier work, in this paper we focus on systems presented in *canonical format*. This format, introduced by the "CMU School", (Watkins *et al.* 2002; Harper and Licata 2007), makes use only of terms in normal form, and equality rules are replaced by *hereditary substitution*. This streamlines the proofs of "adequacy theorems" of the encodings of the object systems in applications. The canonical format is usually rigidly "syntax directed" and it produces a unique derivation for each derivable judgement. In our case this does not hold any longer, but we still have a weak form of syntax directedness that allows for decidability. Uniqueness of derivations is lost, but *inversions* of a derivation can still be performed.

In this paper, first, we discuss the canonical system $\mathsf{CLLF}_{\mathcal{P}}$ and the correspondence to its non-canonical counterpart $\mathsf{LLF}_{\mathcal{P}}$ presented in (Honsell *et al.* 2016).

The second, and completely innovative, contribution of this paper is that of showing that locks can delegate to external tools not only the task of producing suitable evidence but also that of producing suitable *witnesses*, to be further used by the calling environment. This feature is exhibited by the system $\mathsf{CLLF}_{\mathcal{P}?}$ (see Section 3).

In (Honsell *et al.* 2016) 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 when presenting systems in canonical format as we do 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 $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]$ of type $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, via the *type checking rule* ($O{\cdot}Lock$). Correspondingly, we introduce an *unlock destructor*, $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]$, for building atomic objects and an *atomic rule* ($O{\cdot}Unlock$), allowing the elimination, in the hereditary substitution rules, of the lock-type constructor under the condition that a specific predicate $\mathcal{P}$ is verified, possibly *externally*, on a judgement. The specific rules taken from Figures 2, 4, and 5 appear below. We refer to the following sections for the

full explanation of the notation:

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

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

Capitalizing on the monadic nature of the lock constructor, as we did for the systems in (Honsell *et al.* 2014; Honsell *et al.* 2016), one can use locked terms without necessarily first establishing the predicate, provided an *outermost* lock is present. This increases the flexibility of the system, and allows for reasoning under the assumption that the verification is successful, as well as for postponing the test and hence reducing the number of verifications. The rules which make all this work are:

$$\frac{\Gamma, x : \tau \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho]\ \mathsf{type} \quad \Gamma \vdash_\Sigma A \Rightarrow \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau] \quad \rho[\mathcal{U}^{\mathcal{P}}_{S,\sigma}[A]/x]^{\mathcal{F}}_{(\tau)^-} = \rho' \quad x \in \mathsf{Fv}(\rho)}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho']\ \mathsf{type}}\ (F{\cdot}Nested{\cdot}Unlock)$$

$$\frac{\begin{array}{ll}\Gamma, x{:}\tau \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{S}[M] \Leftarrow \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho] & x \in \mathsf{Fv}(M) \cup \mathsf{Fv}(\rho) \\ \Gamma \vdash_\Sigma A \Rightarrow \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau] & \rho[\mathcal{U}^{\mathcal{P}}_{S}[A]/x]^{\mathcal{F}}_{(\tau)^-} = \rho' \quad M[\mathcal{U}^{\mathcal{P}}_{S}[A]/x]^{\mathcal{O}}_{(\tau)^-} = M'\end{array}}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{S}[M'] \Leftarrow \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho']}\ (O{\cdot}Nested{\cdot}Unlock)$$

The (*O·Nested· Unlock*)-rule is the counterpart of the elimination rule for monads, once we realize that the standard destructor of monads (cf. (Moggi 1989)) $let_{T_{\mathcal{P}(\Gamma\vdash S:\sigma)}} x = A\ in\ N$ can be replaced, in our context, by $N[\mathcal{U}^{\mathcal{P}}_{S}[A]/x]$. This holds since the $\mathcal{L}^{\mathcal{P}}_{S}[\cdot]$-monad satisfies the property $let_{T_{\mathcal{P}}}\ x = A\ in\ N \to N[\mathcal{U}^{\mathcal{P}}_{S}[A]/x]$ provided x occurs *guarded* in N, *i.e.* within some subterm whose type is locked by $\mathcal{L}^{\mathcal{P}}_{S}[\cdot]$. Namely, we do not need to check repeatedly the constraint but we do need to do it *at least once*. The rule (*F·Nested· Unlock*) takes care of the elimination rule for monads at the level of types.

The fact that we can express the effects of the elimination rule for monads directly, makes us do away with the tedious *permutative reductions* which normally arise in dealing with monadic *let* constructors. It is precisely the monadic flavour of the above rules, however, that breaks the strict syntax directedness of $\mathsf{CLLF}_{\mathcal{P}}$, which can be recovered in a weaker form sufficient for all practical purposes.

In the second part of the paper, we introduce $\mathsf{CLLF}_{\mathcal{P}?}$. The lock constructor is generalized here to express also the *request* to an external tool for a witness satisfying a given property. If the external search is successful, the unlock operator will then supply it. In $\mathsf{CLLF}_{\mathcal{P}?}$, locks act as *binding operators* while unlocks amount to *substitutions*.

To illustrate the expressive power of $\mathsf{CLLF}_{\mathcal{P}}$ and $\mathsf{CLLF}_{\mathcal{P}?}$ we discuss various challenging encodings of subtle logical systems, as well as some novel applications. First, we encode in $\mathsf{CLLF}_{\mathcal{P}}$ Fitch-Prawitz consistent Set-Theory (FPST), as presented in (Prawitz 1965),

$$
\begin{array}{rcl@{\quad}rcl@{\quad}l}
K & \in & \mathcal{K} & K & ::= & \mathsf{type} \mid \Pi x{:}\sigma.K & \textit{Kinds} \\
\alpha & \in & \mathcal{F}_a & \alpha & ::= & a \mid \alpha\, N & \textit{Atomic Families} \\
\sigma,\tau,\rho & \in & \mathcal{F} & \sigma & ::= & \alpha \mid \Pi x{:}\sigma.\tau \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] & \textit{Canonical Families} \\
A & \in & \mathcal{O}_a & A & ::= & c \mid x \mid A\, M \mid \mathcal{U}^{\mathcal{P}}_{N}[A] & \textit{Atomic Objects} \\
M,N & \in & \mathcal{O} & M & ::= & A \mid \lambda x.M \mid \mathcal{L}^{\mathcal{P}}_{N}[M] & \textit{Canonical Objects} \\
\Sigma & \in & \mathcal{S} & \Sigma & ::= & \emptyset \mid \Sigma, a{:}K \mid \Sigma, c{:}\sigma & \textit{Signatures} \\
\Gamma & \in & \mathcal{C} & \Gamma & ::= & \emptyset \mid \Gamma, x{:}\sigma & \textit{Contexts}
\end{array}
$$

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

and we illustrate its expressive power by showing, by way of example, how it can type all strongly normalizing terms. Then, we give signatures in $\mathsf{CLLF}_{\mathcal{P}}$ of a weakly normalizing λ-calculus and a system of Light Linear Logic (Baillot *et al.* 2007). Finally, in Subsection 4.5, we show how to encode *partial* functions in $\mathsf{CLLF}_{\mathcal{P}?}$.

The paper is organized as follows: in Section 2 we present the syntax, the type system and the metatheory of $\mathsf{CLLF}_{\mathcal{P}}$, whereas $\mathsf{CLLF}_{\mathcal{P}?}$ is introduced in Section 3. Section 4 is devoted to the presentation and discussion of case studies. Finally, connections with related work in the literature appear in Section 5.

2. The Canonical System $\mathsf{CLLF}_{\mathcal{P}}$

In this section, we discuss the system $\mathsf{CLLF}_{\mathcal{P}}$ which is the *canonical* counterpart of $\mathsf{LLF}_{\mathcal{P}}$(Honsell *et al.* 2016) in the style of (Watkins *et al.* 2002; Harper and Licata 2007). This approach amounts to restricting the language only to terms in *long* $\beta\eta\mathcal{UL}$*-normal form*, see Definition 2.4. These are the terms of $\mathsf{LLF}_{\mathcal{P}}$ which are normal w.r.t.

$$
\begin{array}{rcll}
(\lambda x.M)N & \to & M[N/x] & \beta\text{-rule} \\
\mathcal{U}^{\mathcal{P}}_{S}[\mathcal{L}^{\mathcal{P}}_{S}[M]] & \to & M & \mathcal{U}\text{-rule}
\end{array}
$$

and also with respect to typed η-like expansion rules, namely $M \to \lambda x{:}\sigma.Mx$ and $M \to \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]]$, if M is atomic. The added value of canonical systems such as $\mathsf{CLLF}_{\mathcal{P}}$ is that one can streamline results of adequacy for encoded systems. Indeed, terms of the proof meta-language which are not in normal form are not that meaningful. They reflect the use of some, possibly higher-order, derivable rule, which is quite rare in practice.

2.1. *Syntax and Type System for* $\mathsf{CLLF}_{\mathcal{P}}$

The syntax of $\mathsf{CLLF}_{\mathcal{P}}$ is presented in Figure 1. The type system for $\mathsf{CLLF}_{\mathcal{P}}$ is shown in Figure 2. The judgements of $\mathsf{CLLF}_{\mathcal{P}}$ are the following:

$$
\begin{array}{rcll}
 & \Sigma & \mathsf{sig} & \Sigma \text{ is a valid signature} \\
 & \vdash_{\Sigma} & \Gamma & \Gamma \text{ is a valid context in } \Sigma \\
\Gamma & \vdash_{\Sigma} & K & K \text{ is a kind in } \Gamma \text{ and } \Sigma \\
\Gamma & \vdash_{\Sigma} & \sigma\ \mathsf{type} & \sigma \text{ is a canonical family in } \Gamma \text{ and } \Sigma \\
\Gamma & \vdash_{\Sigma} & \alpha \Rightarrow K & K \text{ is the kind of the atomic family } \alpha \text{ in } \Gamma \text{ and } \Sigma \\
\Gamma & \vdash_{\Sigma} & M \Leftarrow \sigma & M \text{ is a canonical term of type } \sigma \text{ in } \Gamma \text{ and } \Sigma \\
\Gamma & \vdash_{\Sigma} & A \Rightarrow \sigma & \sigma \text{ is the type of the atomic term } A \text{ in } \Gamma \text{ and } \Sigma
\end{array}
$$

Valid signatures

$$\frac{}{\emptyset\ \mathsf{sig}}\ (S{\cdot}Empty) \qquad \frac{\Sigma\ \mathsf{sig} \quad \vdash_\Sigma K \quad a \notin \mathsf{Dom}(\Sigma)}{\Sigma, a{:}K\ \mathsf{sig}}\ (S{\cdot}Kind) \qquad \frac{\Sigma\ \mathsf{sig} \quad \vdash_\Sigma \sigma\ \mathsf{type} \quad c \notin \mathsf{Dom}(\Sigma)}{\Sigma, c{:}\sigma\ \mathsf{sig}}\ (S{\cdot}Type)$$

Kind rules

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

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

Atomic Family rules

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

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

Canonical Family rules

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

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

$$\frac{\begin{array}{l}\Gamma \vdash_\Sigma \rho\ \mathsf{type} \\ \Gamma \vdash_\Sigma N \Leftarrow \sigma\end{array}}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]\ \mathsf{type}}\ (F{\cdot}Lock)$$

$$\frac{\begin{array}{l}(F{\cdot}Nested{\cdot}Unlock) \\ \Gamma, x : \tau \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho]\ \mathsf{type} \\ \Gamma \vdash_\Sigma A \Rightarrow \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau] \\ \rho[\mathcal{U}^{\mathcal{P}}_{S,\sigma}[A]/x]^{\mathcal{F}}_{(\tau)^-} = \rho' \quad x \in \mathsf{Fv}(\rho)\end{array}}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho']\ \mathsf{type}}$$

Context rules

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

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

Atomic Object rules

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

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

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

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

Canonical Object rules

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

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

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

$$\frac{\begin{array}{l}\Gamma, x{:}\tau \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_S[M] \Leftarrow \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho] \quad x \in \mathsf{Fv}(M) \cup \mathsf{Fv}(\rho) \\ \Gamma \vdash_\Sigma A \Rightarrow \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau] \\ \rho[\mathcal{U}^{\mathcal{P}}_S[A]/x]^{\mathcal{F}}_{(\tau)^-} = \rho' \qquad M[\mathcal{U}^{\mathcal{P}}_S[A]/x]^{\mathcal{O}}_{(\tau)^-} = M'\end{array}}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_S[M'] \Leftarrow \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho']}\ (O{\cdot}Nested{\cdot}Unlock)$$

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

The judgements $\Sigma\ \mathsf{sig}$, and $\vdash_\Sigma \Gamma$, and $\Gamma \vdash_\Sigma K$ are as in Section 2.1 of (Honsell *et al.* 2013), whereas the remaining ones are peculiar to the canonical style. Informally, the judgment $\Gamma \vdash_\Sigma M \Leftarrow \sigma$ uses σ to *check* the type of the canonical term M, while the judgment $\Gamma \vdash_\Sigma A \Rightarrow \sigma$ uses the type information contained in the atomic term A and

Γ to *synthesize* σ. Without loss of generality, predicates $\mathcal{P}$ in $\mathsf{CLLF}_{\mathcal{P}}$ are defined *only* on judgements of the shape $\Gamma \vdash_\Sigma M \Leftarrow \sigma$.

The type system makes use, in the rules $(A{\cdot}App)$ and $(F{\cdot}App)$, and $(O{\cdot}Nested{\cdot}Unlock)$ and $(F{\cdot}Nested{\cdot}Unlock)$ of the notion of *Hereditary Substitution*, defined in Figures 4 and 5. Hereditary Substitution computes the substitution of one normal form into another, performing β-reductions and Unlock-Lock reductions, *i.e.* $\mathcal{U}^{\mathcal{P}}_S[\mathcal{L}^{\mathcal{P}}_S[M]] \to M$, in the rules $(\mathcal{S}{\cdot}O{\cdot}App{\cdot}H)$ and $(\mathcal{S}{\cdot}O{\cdot}Unlock{\cdot}H)$, when a redex would result from the substitution. The general form of the hereditary substitution judgement is $T[M/x]^t_\rho = T'$, 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 $t \in \{\mathcal{K}, \mathcal{F}, \mathcal{F}_a, \mathcal{O}, \mathcal{O}_a\}$ denotes the syntactic class (*i.e.*, kinds, canonical families/objects, atomic families/object) under consideration. We give the rules of the Hereditary Substitution in the style of (Harper and Licata 2007), where the erasure function to simple types is necessary to ensure the decidability of the existence of a hereditary substitution even in presence of ill-formed terms (cf. (Harper and Licata 2007)).

The simple-type ρ of M is obtained via the erasure function of (Harper and Licata 2007) (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 $(O{\cdot}Atom)$ of the type system (Figure 2), the syntactic restriction of the classifier to α atomic ensures that canonical forms are *long $\beta\eta\mathcal{UL}$-normal forms* for the appropriate notion of long $\beta\eta$-normal form, which extends the standard one to lock-types (Definition 2.4). Hence, since 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, then $\vdash_\Sigma \lambda x.x \Leftarrow \Pi x{:}(\Pi z{:}a.a).\Pi z{:}a.a$ is not derivable either. On the other hand, $\vdash_\Sigma \lambda x.\lambda y.xy \Leftarrow \Pi x{:}(\Pi z{:}a.a).\,\Pi z{:}a.a$, 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. Consequently, $\vdash_\Sigma \lambda x.x \Leftarrow \Pi x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho].\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$ is not derivable either. However, $x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{U}^{\mathcal{P}}_N[x]] \Leftarrow \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$ is derivable, if ρ is atomic. Hence, the judgment $\vdash_\Sigma \lambda x.\mathcal{L}^{\mathcal{P}}_N[\mathcal{U}^{\mathcal{P}}_N[x]] \Leftarrow \Pi x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho].\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$ is derivable. Notice that the unlock constructor takes an *atomic* term as its main argument, thus avoiding the creation of possible $\mathcal{L}$-redexes under substitution. Moreover, as unlocks can only receive locked terms in their body, no abstractions can ever arise. In Definition 2.4, we formalize the notion of η-expansion of a judgement, together with correspondence theorems between $\mathsf{LLF}_{\mathcal{P}}$ and $\mathsf{CLLF}_{\mathcal{P}}$.

Because of the monadic behaviour of locks, there are two rules, whose conclusion involves the lock constructor, $\mathcal{L}^{\mathcal{P}}_S[\cdot]$, which apparently break syntax directedness, namely $(F{\cdot}Nested{\cdot}Unlock)$ and $(O{\cdot}Nested{\cdot}Unlock)$ (see Figure 2). Syntax directedness fails in two possible ways. In the first place, we need to know when to choose them versus the corresponding *un-nested* rules, namely $(F{\cdot}Unlock)$ and $(O{\cdot}Unlock)$. This can be easily solved by applying the un-nested rules only when M and ρ are $\mathcal{U}^{\mathcal{P}}_N[\cdot]$-free, *i.e.* neither M nor ρ contain subterms of the shape $\mathcal{U}^{\mathcal{P}}_N[\cdot]$. This amounts to avoiding unnecessary locks once we already know that the lock is satisfied. The other possible source of undeterminacy derives from the fact that in both rules the argument of the $\mathcal{U}^{\mathcal{P}}_S[A]$, *i.e.* A, might not be uniquely determined. However any A, say the *leftmost* in M', satisfying the proviso to be "$\mathcal{U}^{\mathcal{P}}_N[\cdot]$-free" gives a principled strategy which is deterministic for all practical pur-

$$\frac{}{(a)^- = a} \qquad \frac{(\alpha)^- = \rho}{(\alpha\, M)^- = \rho} \qquad \frac{(\sigma)^- = \rho_1 \quad (\tau)^- = \rho_2}{(\Pi x{:}\sigma.\tau)^- = \rho_1 \to \rho_2} \qquad \frac{(\tau)^- = \rho}{(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\tau])^- = \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]}$$

Fig. 3. Erasure to simple-types

Substitution in Kinds

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

Substitution in Atomic Families

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

Substitution in Canonical Families

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

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

Fig. 4. Hereditary substitution, kinds and families of $\mathsf{CLLF}_{\mathcal{P}}$

poses. Notice that for this to work, it is necessary that Hereditary Substitution behaves as a standard substitution when an argument of the shape $\mathcal{U}^{\mathcal{P}}_{S}[A]$ is supplied to it. These criteria in using the rules $(F{\cdot}Nested{\cdot}Unlock)$ and $(O{\cdot}Nested{\cdot}Unlock)$ are a somewhat weaker form of syntax directedness, which nevertheless ensures decidability. We could have directly included the provisos in the rule of the system $\mathsf{CLLF}_{\mathcal{P}}$ in Figure 2, but this would have made the Correspondence Theorem 2.9 opaque.

In this paper, we present $\mathsf{CLLF}_{\mathcal{P}}$ *à la* Curry, following closely (Harper and Licata 2007), while in (Honsell *et al.* 2016) we presented the standard (*i.e.* non canonical) system $\mathsf{LLF}_{\mathcal{P}}$ in a fully-typed style, *i.e. à la* Church. Namely, in $\mathsf{LLF}_{\mathcal{P}}$ the canonical forms $\lambda x.M$, $\mathcal{L}^{\mathcal{P}}_{M}[N]$, and $\mathcal{U}^{\mathcal{P}}_{M}[N]$ carry type information. We could have made that choice also for $\mathsf{CLLF}_{\mathcal{P}}$ (as we did in (Honsell *et al.* 2015)), the type rules would then have been, *e.g.*:

$$\frac{\Gamma, x{:}\sigma \vdash_{\Sigma} M \Leftarrow \tau}{\Gamma \vdash_{\Sigma} \lambda x{:}\sigma.M \Leftarrow \Pi x{:}\sigma.\tau}\ (O{\cdot}Abs) \qquad \frac{\Gamma \vdash_{\Sigma} M \Leftarrow \sigma \quad \Gamma \vdash_{\Sigma} N \Leftarrow \tau}{\Gamma \vdash_{\Sigma} \mathcal{L}^{\mathcal{P}}_{M,\sigma}[N] \Leftarrow \mathcal{L}^{\mathcal{P}}_{M,\sigma}[\tau]}\ (O{\cdot}Lock)$$

The Curry syntax is more suitable in implementations because it simplifies the notation, while, as remarked in (Honsell *et al.* 2015), the typeful syntax *à la* Church allows for a more direct comparison with non-canonical systems. This, however, is technically immaterial. In the Correspondence Theorem 2.9 we prove 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. And 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

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

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

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

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

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

Substitution in Canonical Objects

$$\frac{A[M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = A'}{A[M_0/x_0]^{\mathcal{O}}_{\rho_0} = A'}\,(\mathcal{S}{\cdot}O{\cdot}R) \quad \frac{A[M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = M' : \rho}{A[M_0/x_0]^{\mathcal{O}}_{\rho_0} = M'}\,(\mathcal{S}{\cdot}O{\cdot}R{\cdot}H) \quad \frac{M[M_0/x_0]^{\mathcal{O}}_{\rho_0} = M'}{\lambda x.M[M_0/x_0]^{\mathcal{O}}_{\rho_0} = \lambda x.M'}\,(\mathcal{S}{\cdot}O{\cdot}Abs)$$

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

Substitution in Contexts

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

Fig. 5. Hereditary substitution, objects and contexts of $\mathsf{CLLF}_{\mathcal{P}}$

2.2. *The Metatheory of* $\mathsf{CLLF}_{\mathcal{P}}$

The type system $\mathsf{CLLF}_{\mathcal{P}}$ is legitimately a Logical Framework in that it is decidable (Theorem 2.6), provided the predicates $\mathcal{P}$ are themselves decidable. Moreover, it captures the expressive power of $\mathsf{LLF}_{\mathcal{P}}$ in the sense of the Soundness and Correspondence Theorems 2.8 and 2.9. Soundness implies that every valid judgement J in $\mathsf{CLLF}_{\mathcal{P}}$ is, up to unique type decoration of λ's and lock/unlocks, a valid judgement in $\mathsf{LLF}_{\mathcal{P}}$. Correspondence claims that a judgement J is valid in $\mathsf{LLF}_{\mathcal{P}}$ if and only if there exists a valid judgment J' in $\mathsf{CLLF}_{\mathcal{P}}$ which morally is the long $\eta\mathcal{L}$-expansions of the $\beta\mathcal{U}$-normal forms of all its components. We will clarify this point when discussing the Theorems, without spelling out all the details because they belong to the *Logical Frameworks' folklore*.

Throughout this section we capitalize, whenever possible, on the seminal work (Harper and Licata 2007) and the canonical version of the system introduced in (Honsell *et al.* 2013). Indeed, all the proofs follow the standard patterns used in those papers. The only remark-worthy differences w.r.t. the approach of (Harper and Licata 2007) are the need to take care of the lock and unlock constructors and the fact that we drop the subordination relation from the typing system, taking the strongest one as implicit (for the details, see Section 2.4 of (Harper and Licata 2007)).

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 Definition 1 (Honsell *et al.* 2013). 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 $\mathcal{P}$ in the set satisfies the following conditions, provided all the judgements involved are valid:

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

2 ***Closure under hereditary substitution:*** If $\mathcal{P}(\Gamma, x{:}\sigma', \Gamma' \vdash_\Sigma N \Leftarrow \sigma)$ and $\Gamma \vdash_\Sigma N' \Leftarrow \sigma'$, then $\mathcal{P}(\Gamma, \Gamma'[N'/x]^{\mathcal{C}}_{(\sigma')^-} \vdash_\Sigma N[N'/x]^{\mathcal{O}}_{(\sigma')^-} \Leftarrow \sigma[N'/x]^{\mathcal{F}}_{(\sigma')^-})$.

As canonical systems do not feature reductions, 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 (Head substitution size).
If $A[M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = M{:}\rho$, then ρ is a subexpression of ρ_0.

Proof. The proof proceeds by induction on the derivation of $A[M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = M{:}\rho$. Hence, the involved rules are $(\mathcal{S}{\cdot}O{\cdot}Var{\cdot}H)$, $(\mathcal{S}{\cdot}O{\cdot}App{\cdot}H)$, and $(\mathcal{S}{\cdot}O{\cdot}Unlock{\cdot}H)$. The only difference w.r.t. (Harper and Licata 2007) is represented by the latter case, where, by induction hypothesis, we have that $\mathcal{L}^{\mathcal{P}}_{M',\sigma'}[\rho]$ is a subexpression of ρ_0, whence also ρ is a subexpression of ρ_0. □

Lemma 2.2 (Uniqueness of substitution and synthesis).

1 It is not possible that $A[M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = A'$ and $A[M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = M{:}\rho$.
2 For any T in any category $t \in \{\mathcal{K}, \mathcal{F}_a, \mathcal{F}, \mathcal{O}_a, \mathcal{O}\}$, if $T[M_0/x_0]^t_{\rho_0} = T'$, and $T[M_0/x_0]^t_{\rho_0} = T''$, then $T' = T''$.
3 If $\Gamma \vdash_\Sigma A \Rightarrow \sigma$, and $\Gamma \vdash_\Sigma A \Rightarrow \sigma'$, then $\sigma = \sigma'$.
4 If $\Gamma \vdash_\Sigma \alpha \Rightarrow K$, and $\Gamma \vdash_\Sigma \alpha \Rightarrow K'$, then $K = K'$.

Proof. these claims follow directly from the definition of hereditary substitution (see Figures 4 and 5) and the $\mathsf{CLLF}_{\mathcal{P}}$ type system (see Figure 2). □

So far, we can state and prove the following lemma about the decidability of hereditary substitution:

Lemma 2.3 (Decidability of hereditary substitution).

1 For any T in $\{\mathcal{K}, \mathcal{F}_a, \mathcal{F}, \mathcal{O}, \mathcal{C}\}$, and any M, x, and ρ, it is decidable whether there exists a T' such that $T[M/x]^m_\rho = T'$ or there is no such T'.

2 For any M, x, ρ, and A, it is decidable whether there exists an A', such that $A[M/x]^{\mathcal{O}_a}_{\rho} = A'$, or there exist M' and ρ', such that $A[M/x]^{\mathcal{O}_a}_{\rho} = M' : \rho'$, or there are no such A' and M'.

Proof. This is the lemma for which *erasure to simple types* (Figure 3) plays a crucial role. As in (Harper and Licata 2007), the proof proceeds first with a mutual lexicographic induction on the simple type ρ, the terms M and A, and an order allowing inductive calls to clauses for atomic terms from clauses of canonical terms (when, of course, the terms and the simple types involved are the same), in order to prove claim 1 for canonical terms (*i.e.*, $T \in \mathcal{O}$) and claim 2. Then, another induction on M is carried out to prove the remaining clauses of the first part about $\mathcal{F}$, $\mathcal{F}_a$, $\mathcal{K}$, and $\mathcal{C}$. □

Lemma 2.4 (Composition of hereditary substitution). Let $x \neq x_0$ and $x \notin \mathsf{Fv}(M_0)$. Then:

1 For all T_1' in $\{\mathcal{K}, \mathcal{F}_a, \mathcal{F}, \mathcal{O}_a, \mathcal{O}\}$, if we have that $M_2[M_0/x_0]^{\mathcal{O}}_{\rho_0} = M_2'$, $T_1[M_2/x]^{m}_{\rho_2} = T_1'$, and $T_1[M_0/x_0]^{m}_{\rho_0} = T_1''$, then there exists a T: $T_1'[M_0/x_0]^{m}_{\rho_0} = T$, and $T_1''[M_2'/x]^{m}_{\rho_2} = T$.
2 If $M_2[M_0/x_0]^{\mathcal{O}}_{\rho_0} = M_2'$, $A_1[M_2/x]^{\mathcal{O}_a}_{\rho_2} = M : \rho$, and $A_1[M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = A$, then there exists an M': $M[M_0/x_0]^{\mathcal{O}}_{\rho_0} = M'$, and $A[M_2'/x]^{\mathcal{O}_a}_{\rho_2} = M' : \rho$.
3 If $M_2[M_0/x_0]^{\mathcal{O}}_{\rho_0} = M_2'$, $A_1[M_2/x]^{\mathcal{O}_a}_{\rho_2} = A$, and $A_1[M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = M : \rho$, then there exists an M': $A[M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = M' : \rho$, and $M[M_2'/x]^{\mathcal{O}}_{\rho_2} = M'$.

Proof. These claims are proved following the same pattern of (Harper and Licata 2007), by means of mutual induction on $size(\rho_0) + size(\rho_2)$ and on the derivation of the substitution of M_2 (where $size(a) = 1$ and $size(\rho_1 \to \rho_2) = 1 + size(\rho_1) + size(\rho_2)$). □

Then, by induction on derivations, similar to one in (Harper and Licata 2007) p.14–15, we can prove:

Theorem 2.5 (Transitivity). Let Σ sig, $\vdash_\Sigma \Gamma, x_0{:}\rho_0, \Gamma'$ and $\Gamma \vdash_\Sigma M_0 \Leftarrow \rho_0$, and assume that all predicates are well-behaved. Then,

1 There exists a Γ'': $\Gamma'[M_0/x_0]^{\mathcal{C}}_{\rho_0} = \Gamma''$ and $\vdash_\Sigma \Gamma, \Gamma''$.
2 If $\Gamma, x_0{:}\rho_0, \Gamma' \vdash_\Sigma K$ then there exists a K': $[M_0/x_0]^{\mathcal{K}}_{\rho_0} K = K'$ and $\Gamma, \Gamma'' \vdash_\Sigma K'$.
3 If $\Gamma, x_0{:}\rho_0, \Gamma' \vdash_\Sigma \sigma$ type, then there exists a σ': $[M_0/x_0]^{\mathcal{F}}_{\rho_0}\sigma = \sigma'$ and $\Gamma, \Gamma'' \vdash_\Sigma \sigma'$ type.
4 If $\Gamma, x_0{:}\rho_0, \Gamma' \vdash_\Sigma \sigma$ type and $\Gamma, x_0{:}\rho_0, \Gamma' \vdash_\Sigma M \Leftarrow \sigma$, then there exist σ' and M': $[M_0/x_0]^{\mathcal{F}}_{\rho_0}\sigma = \sigma'$ and $[M_0/x_0]^{\mathcal{O}}_{\rho_0} M = M'$ and $\Gamma, \Gamma'' \vdash_\Sigma M' \Leftarrow \sigma'$.

Theorem 2.6 (Decidability of typing). If predicates in $\mathsf{CLLF}_{\mathcal{P}}$ are decidable, then all of the judgements of the system are decidable.

Proof. By induction on the complexity of judgements we reconstruct derivations when possible. The only cases which differ from the proof in (Honsell *et al.* 2013) are those regarding the rules ($F{\cdot}Nested{\cdot}Unlock$) and ($O{\cdot}Nested{\cdot}Unlock$), and ($F{\cdot}Lock$) and ($O{\cdot}Lock$) (see Figure 2). We proceed as outlined before in discussing the weak form of syntax directedness of the system. Namely rules ($F{\cdot}Lock$) and ($O{\cdot}Lock$) are inverted only when the arguments to the locks are $\mathcal{U}^{\mathcal{P}}_{S}[\cdot]$-free. This excludes the possibility of reconstructing derivations which lock terms with the predicate $\mathcal{P}$ which have subterms of the shape $\mathcal{U}^{\mathcal{P}}_{S}[\cdot]$, when we already know the predicate $\mathcal{P}$ to hold, *i.e.* we have used already rule

($O{\cdot}Unlock$). It is immediate to check that such proofs do not extend the class of derivable judgments. Similarly, when we have to choose a suitable A in rules ($F{\cdot}Nested{\cdot}Unlock$) and ($O{\cdot}Nested{\cdot}Unlock$), we inspect M' searching for all the occurrences of subterms of the shape $\mathcal{U}^{\mathcal{P}}_S[A]$, and we choose any A which is itself $\mathcal{U}^{\mathcal{P}}_S[\cdot]$-free, say the *leftmost* in M'. This provides a principled deterministic procedure, which does not require any backtracking. It is tedious but straightforward to check that the proofs which are excluded do not extend the class of derivable judgments. □

Notice that in case the predicates $\mathcal{P}$ are only *semidecidable* then the same argument in the above proof yields that the system $\mathsf{CLLF}_{\mathcal{P}}$ is semidecidable.

We can now state precisely the relationship between $\mathsf{CLLF}_{\mathcal{P}}$ and the system $\mathsf{LLF}_{\mathcal{P}}$ in (Honsell *et al.* 2013). We assume that the reader is familiar with (Honsell *et al.* 2013). First, we need to introduce the natural erasure function $\mathcal{E}$ which removes types from λ abstractions, and turns terms of the form $\mathcal{L}^{\mathcal{P}}_{S,\sigma}[M]$ into $\mathcal{L}^{\mathcal{P}}_S[M]$ and $\mathcal{U}^{\mathcal{P}}_{S,\sigma}[M]$ into $\mathcal{U}^{\mathcal{P}}_S[M]$. The function $\mathcal{E}$ is defined in the obvious way and extends to all syntactic categories.

Definition 2.2 (Erasure function). The erasure function $\mathcal{E}$ maps terms of $\mathsf{LLF}_{\mathcal{P}}$ (in Church-style) into the corresponding terms of $\mathsf{CLLF}_{\mathcal{P}}$ (in Curry-style) as follows:

$$\begin{array}{lcl}
\mathcal{E}(c) & = & c \\
\mathcal{E}(x) & = & x \\
\mathcal{E}(MN) & = & \mathcal{E}(M)\mathcal{E}(N) \\
\mathcal{E}(\lambda x{:}\sigma.M) & = & \lambda x.\mathcal{E}(M) \\
\mathcal{E}(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]) & = & \mathcal{L}^{\mathcal{P}}_{\mathcal{E}(N)}[\mathcal{E}(M)] \\
\mathcal{E}(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]) & = & \mathcal{U}^{\mathcal{P}}_{\mathcal{E}(N)}[\mathcal{E}(M)]
\end{array}$$

Then, $\mathcal{E}$ is naturally extended to type families, kinds, signatures and contexts as follows:

$$\begin{array}{lcl}
\mathcal{E}(a) & = & a \\
\mathcal{E}(\Pi x{:}\sigma.\tau) & = & \Pi x{:}\mathcal{E}(\sigma).\mathcal{E}(\tau) \\
\mathcal{E}(\sigma N) & = & \mathcal{E}(\sigma)\mathcal{E}(N) \\
\mathcal{E}(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\tau]) & = & \mathcal{L}^{\mathcal{P}}_{\mathcal{E}(N),\mathcal{E}(\sigma)}[\mathcal{E}(\tau)] \\
\mathcal{E}(\mathsf{type}) & = & \mathsf{type} \\
\mathcal{E}(\Pi x{:}\sigma.K) & = & \Pi x{:}\mathcal{E}(\sigma).\mathcal{E}(K) \\
\mathcal{E}(\Sigma, a{:}K) & = & \mathcal{E}(\Sigma), a{:}\mathcal{E}(K) \\
\mathcal{E}(\Sigma, c{:}\sigma) & = & \mathcal{E}(\Sigma), c{:}\mathcal{E}(\sigma) \\
\mathcal{E}(\Gamma, x{:}\sigma) & = & \mathcal{E}(\Gamma), x{:}\mathcal{E}(\sigma)
\end{array}$$

Next, we introduce the crucial notion of a judgement in *long* $\beta\eta\mathcal{UL}$*-normal form* ($\beta\eta\mathcal{UL}$-lnf).

Definition 2.3. An occurrence ξ of a constant or a variable in a term of an $\mathsf{LLF}_{\mathcal{P}}$ judgement is *fully applied and unlocked* w.r.t. its type or kind $\Pi\vec{x}_1{:}\vec{\sigma}_1.\vec{\mathcal{L}}_1[\ldots\Pi\vec{x}_n{:}\vec{\sigma}_n.\vec{\mathcal{L}}_n[\alpha]\ldots]$, where $\vec{\mathcal{L}}_1,\ldots,\vec{\mathcal{L}}_n$ are vectors of locks, if ξ appears only in contexts that are of the form $\vec{\mathcal{U}}_n[(\ldots(\vec{\mathcal{U}}_1[\xi\vec{M}_1])\ldots)\vec{M}_n]$, where $\vec{M}_1,\ldots,\vec{M}_n$, $\vec{\mathcal{U}}_1,\ldots,\vec{\mathcal{U}}_n$ have the same arities of the corresponding vectors of Π's and locks.

Definition 2.4 (Judgements in long $\beta\eta\mathcal{UL}$-normal form).

1 A term T in a judgement of $\mathsf{LLF}_{\mathcal{P}}$ is in *$\beta\eta\mathcal{UL}$-lnf* if T is in normal form and every constant and variable occurrence in T is fully applied and unlocked w.r.t. its typing in the judgement.
2 A judgement is in *$\beta\eta\mathcal{UL}$-lnf* if all terms appearing in it are in $\beta\eta\mathcal{UL}$-lnf.

The concept of a "judgement in $\beta\eta\mathcal{UL}$-lnf" is based on the idea of inverting the standard η-rule. This is made precise in the following theorem. First we introduce the following two notions of reduction:

$M \to_{\eta_{long}} \lambda x{:}\sigma.Mx$ provided both M and $\lambda x{:}\sigma.Mx$ are in β-normal form;
$M \to_{\mathcal{L}_{long}} \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]]$ provided M and $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]]$ are in $\mathcal{UL}$-normal form.

Theorem 2.7. Let J be a valid judgement in $\mathsf{LLF}_{\mathcal{P}}$. Then, there exists a unique valid judgement $J^\sharp$ in $\beta\eta\mathcal{UL}$-lnf such that all terms appearing in $J^\sharp$ are $\beta\mathcal{U}\eta_{long}\mathcal{L}_{long}$-reducts of the corresponding terms in J.

Theorem 2.7 above is proved by induction on the derivation of judgments in $\beta\mathcal{UL}$-normal form in $\mathsf{LLF}_{\mathcal{P}}$, once the full power of Subject Reduction in $\mathsf{LLF}_{\mathcal{P}}$ has been used to obtain the "normal form" of a judgement J. Actually, Theorem 2.7 yields a function which maps each term T of a valid judgment to its "$\beta\mathcal{U}\eta_{long}\mathcal{L}_{long}$-normal form" $T^\sharp$.

We are now ready to prove the two fundamental theorems:

Theorem 2.8 (Soundness). For any well-behaved predicate $\mathcal{P}$ of $\mathsf{CLLF}_{\mathcal{P}}$, we define a corresponding predicate $\mathcal{P}'$ in $\mathsf{LLF}_{\mathcal{P}}$ as follows: $\mathcal{P}'(\Gamma \vdash_\Sigma M : \sigma)$ holds if and only if $\Gamma \vdash_\Sigma M : \sigma$ is derivable in $\mathsf{LLF}_{\mathcal{P}}$ and $\mathcal{P}(\mathcal{E}(\Gamma^\sharp) \vdash_{\mathcal{E}(\Sigma^\sharp)} \mathcal{E}(M^\sharp) \Leftarrow \mathcal{E}(\sigma^\sharp))$ holds in $\mathsf{CLLF}_{\mathcal{P}}$. Then we have:

1 If Σ sig is $\mathsf{CLLF}_{\mathcal{P}}$-derivable, then there exists a unique Σ', such that Σ' sig is $\mathsf{LLF}_{\mathcal{P}}$-derivable and $\mathcal{E}(\Sigma') = \Sigma$.
2 If $\vdash_\Sigma \Gamma$ is $\mathsf{CLLF}_{\mathcal{P}}$-derivable, then there exist unique Σ', Γ', such that $\vdash_{\Sigma'} \Gamma'$ is $\mathsf{LLF}_{\mathcal{P}}$-derivable, and $\mathcal{E}(\Sigma') = \Sigma$ and $\mathcal{E}(\Gamma') = \Gamma$.
3 If $\Gamma \vdash_\Sigma K$ is $\mathsf{CLLF}_{\mathcal{P}}$-derivable, then there exist unique Σ', Γ' and K', such that $\Gamma' \vdash_{\Sigma'} K'$ is $\mathsf{LLF}_{\mathcal{P}}$-derivable, and $\mathcal{E}(\Sigma') = \Sigma$, $\mathcal{E}(\Gamma') = \Gamma$, and $\mathcal{E}(K') = K$.
4 If $\Gamma \vdash_\Sigma \alpha \Leftarrow K$ is $\mathsf{CLLF}_{\mathcal{P}}$-derivable, then there exist unique Σ', Γ', K', and α', such that $\Gamma' \vdash_{\Sigma'} \alpha' : K'$ is $\mathsf{LLF}_{\mathcal{P}}$-derivable, and $\mathcal{E}(\Sigma') = \Sigma$, $\mathcal{E}(\Gamma') = \Gamma$, $\mathcal{E}(K') = K$, and $\mathcal{E}(\alpha') = \alpha$.
5 If $\Gamma \vdash_\Sigma \sigma$ type is $\mathsf{CLLF}_{\mathcal{P}}$-derivable, then there exist unique Σ', Γ', and σ', such that $\Gamma' \vdash_{\Sigma'} \sigma'$: type is $\mathsf{LLF}_{\mathcal{P}}$-derivable, and $\mathcal{E}(\Sigma') = \Sigma$, $\mathcal{E}(\Gamma') = \Gamma$, and $\mathcal{E}(\sigma') = \sigma$.
6 If $\Gamma \vdash_\Sigma A \Rightarrow \sigma$ is $\mathsf{CLLF}_{\mathcal{P}}$-derivable, then there exist unique Σ', Γ', σ', and A', such that $\Gamma' \vdash_{\Sigma'} A' : \sigma'$ is $\mathsf{LLF}_{\mathcal{P}}$-derivable, and $\mathcal{E}(\Sigma') = \Sigma$, $\mathcal{E}(\Gamma') = \Gamma$, $\mathcal{E}(\sigma') = \sigma$, and $\mathcal{E}(A') = A$.
7 If $\Gamma \vdash_\Sigma M \Leftarrow \sigma$ is $\mathsf{CLLF}_{\mathcal{P}}$-derivable, then there exist unique Σ', Γ', σ', and M', such that $\Gamma' \vdash_{\Sigma'} M' : \sigma'$ is $\mathsf{LLF}_{\mathcal{P}}$-derivable, and $\mathcal{E}(\Sigma') = \Sigma$, $\mathcal{E}(\Gamma') = \Gamma$, $\mathcal{E}(\sigma') = \sigma$, and $\mathcal{E}(M') = M$.

Proof. The proof proceeds by a lengthy, but ultimately straightforward mutual induction on the structure of the $\mathsf{CLLF}_{\mathcal{P}}$ derivations. □

Theorem 2.9 (Correspondence). For any well-behaved predicate $\mathcal{P}$ in $\mathsf{LLF}_{\mathcal{P}}$, in the sense of Definition 1 (Honsell *et al.* 2013) we define a corresponding predicate $\mathcal{P}'$ in $\mathsf{CLLF}_{\mathcal{P}}$ such that $\mathcal{P}'(\mathcal{E}(\Gamma) \vdash_{\mathcal{E}(\Sigma)} \mathcal{E}(M) \Leftarrow \mathcal{E}(\sigma))$ holds if $\mathcal{E}(\Gamma) \vdash_{\mathcal{E}(\Sigma)} \mathcal{E}(M) \Leftarrow \mathcal{E}(\sigma)$ is derivable in $\mathsf{CLLF}_{\mathcal{P}}$ and $\mathcal{P}(\Gamma \vdash_{\Sigma} M : \sigma)$ holds in $\mathsf{LLF}_{\mathcal{P}}$. Then we have:

1 If Σ sig is in $\beta\eta\mathcal{UL}$-lnf and is $\mathsf{LLF}_{\mathcal{P}}$-derivable, then $\mathcal{E}(\Sigma)$ sig is $\mathsf{CLLF}_{\mathcal{P}}$-derivable.
2 If $\vdash_{\Sigma} \Gamma$ is in $\beta\eta\mathcal{UL}$-lnf and is $\mathsf{LLF}_{\mathcal{P}}$-derivable, then $\vdash_{\mathcal{E}(\Sigma)} \mathcal{E}(\Gamma)$ is $\mathsf{CLLF}_{\mathcal{P}}$-derivable.
3 If $\Gamma \vdash_{\Sigma} K$ is in $\beta\eta\mathcal{UL}$-lnf, and is $\mathsf{LLF}_{\mathcal{P}}$-derivable, then $\mathcal{E}(\Gamma) \vdash_{\mathcal{E}(\Sigma)} \mathcal{E}(K)$ is $\mathsf{CLLF}_{\mathcal{P}}$-derivable.
4 If $\Gamma \vdash_{\Sigma} \alpha : K$ is in $\beta\eta\mathcal{UL}$-lnf, except for possibly the head variable/constant of α which is not fully applied, and is $\mathsf{LLF}_{\mathcal{P}}$-derivable, then $\mathcal{E}(\Gamma) \vdash_{\mathcal{E}(\Sigma)} \mathcal{E}(\alpha) \Rightarrow \mathcal{E}(K)$ is $\mathsf{CLLF}_{\mathcal{P}}$-derivable.
5 If $\Gamma \vdash_{\Sigma} \sigma$:type is in $\beta\eta\mathcal{UL}$-lnf and is $\mathsf{LLF}_{\mathcal{P}}$-derivable, then $\mathcal{E}(\Gamma) \vdash_{\mathcal{E}(\Sigma)} \mathcal{E}(\sigma)$ type is $\mathsf{CLLF}_{\mathcal{P}}$-derivable.
6 If $\Gamma \vdash_{\Sigma} A : \alpha$ is in $\beta\eta\mathcal{UL}$-lnf, except for possibly the head variable/constant of A which is not fully applied, and is $\mathsf{LLF}_{\mathcal{P}}$-derivable, then $\mathcal{E}(\Gamma) \vdash_{\mathcal{E}(\Sigma)} \mathcal{E}(A) \Rightarrow \mathcal{E}(\alpha)$ is $\mathsf{CLLF}_{\mathcal{P}}$-derivable.
7 If $\Gamma \vdash_{\Sigma} M : \sigma$ is in $\beta\eta\mathcal{UL}$-lnf and is $\mathsf{LLF}_{\mathcal{P}}$-derivable, then $\mathcal{E}(\Gamma) \vdash_{\mathcal{E}(\Sigma)} \mathcal{E}(M) \Leftarrow \mathcal{E}(\sigma)$ is $\mathsf{CLLF}_{\mathcal{P}}$-derivable.

Proof. The strategy follows closely the one used in the proof of the similar Correspondence Theorem 5.11 in (Honsell *et al.* 2013), where all items are proved 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.

We illustrate in full detail the subcase of point 7, when the $\mathsf{LLF}_{\mathcal{P}}$-derivable judgement is $\Gamma \vdash_{\Sigma} \mathcal{L}^{\mathcal{P}}_{S,\sigma}[M]{:}\theta$. By inspecting the typing rules of $\mathsf{LLF}_{\mathcal{P}}$, we have that $\theta \equiv \mathcal{L}^{\mathcal{P}}_{S',\sigma'}[\rho]$, where $S{=}_{\beta\mathcal{L}}S'$ and $\sigma{=}_{\beta\mathcal{L}}\sigma'$. Moreover, since the judgement is in $\beta\eta\mathcal{UL}$-lnf, we have that $S \equiv S'$ and $\sigma \equiv \sigma'$. Thus, if neither M nor ρ have subterms of the shape $\mathcal{U}^{\mathcal{P}}_{S,\sigma}[\,]$, we can proceed like in the proof of the analogous Correspondence Theorem of (Honsell *et al.* 2013). Otherwise, we have that the original introduction rule for our judgement must be $(O{\cdot}Guarded{\cdot}Unlock)$. In that case, there must be a term M' such that $M \equiv M'[\mathcal{U}^{\mathcal{P}}_{S'',\sigma''}[N]/x]$ and $\rho \equiv \rho'[\mathcal{U}^{\mathcal{P}}_{S'',\sigma''}[N]/x]$ with $S{=}_{\beta\mathcal{L}}S''$ and $\sigma{=}_{\beta\mathcal{L}}\sigma''$, but it must be that $S \equiv S''$ and $\sigma \equiv \sigma''$, since the judgement is in $\beta\eta\mathcal{UL}$-lnf. Hence, we also have that $\Gamma, x{:}\tau \vdash_{\Sigma} \mathcal{L}^{\mathcal{P}}_{S,\sigma}[M']{:}\mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho']$ and $\Gamma \vdash_{\Sigma} N{:}\mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau]$, both judgements in $\beta\eta\mathcal{UL}$-lnf.

By applying the induction hypothesis to the last two judgments, we obtain that $\mathcal{E}(\Gamma), x{:}\mathcal{E}(\tau) \vdash_{\mathcal{E}(\Sigma)} \mathcal{L}^{\mathcal{P}}_{\mathcal{E}(S)}[\mathcal{E}(M')] \Leftarrow \mathcal{L}^{\mathcal{P}}_{\mathcal{E}(S),\mathcal{E}(\sigma)}[\mathcal{E}(\rho')]$ and also that $\mathcal{E}(\Gamma) \vdash_{\mathcal{E}(\Sigma)} \mathcal{E}(N) \Leftarrow \mathcal{L}^{\mathcal{P}}_{\mathcal{E}(S),\mathcal{E}(\sigma)}[\mathcal{E}(\tau)]$ in $\mathsf{CLLF}_{\mathcal{P}}$. Since the latter judgement can only be derived from the rule $(O{\cdot}Atom)$, we also have that $\mathcal{E}(\Gamma) \vdash_{\mathcal{E}(\Sigma)} \mathcal{E}(N) \Rightarrow \mathcal{L}^{\mathcal{P}}_{\mathcal{E}(S),\mathcal{E}(\sigma)}[\mathcal{E}(\tau)]$ holds in $\mathsf{CLLF}_{\mathcal{P}}$. Now we

may apply the rule $(O{\cdot}Nested{\cdot}Unlock)^{\dagger}$ to obtain $\mathcal{E}(\Gamma) \vdash_{\mathcal{E}(\Sigma)} \mathcal{L}^{\mathcal{P}}_{\mathcal{E}(S)}[\mathcal{E}(M')[\mathcal{U}^{\mathcal{P}}_{\mathcal{E}(S)}[\mathcal{E}(N)]/x]] \Leftarrow \mathcal{L}^{\mathcal{P}}_{\mathcal{E}(S),\mathcal{E}(\sigma)}[\mathcal{E}(\rho')[\mathcal{U}^{\mathcal{P}}_{\mathcal{E}(S)}[\mathcal{E}(N)]/x]]$, which is equivalent to $\mathcal{E}(\Gamma) \vdash_{\mathcal{E}(\Sigma)} \mathcal{E}(\mathcal{L}^{\mathcal{P}}_{S}[M]) \Leftarrow \mathcal{E}(\mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho])$, *i.e.*, the thesis. □

So far we do not know yet that the two predicates $\mathcal{P}'$ defined in the above Soundness and Correspondence Theorems are well-behaved predicates in $\mathsf{LLF}_{\mathcal{P}}$ and $\mathsf{CLLF}_{\mathcal{P}}$ respectively. This is established in the following:

Theorem 2.10. The predicate $\mathcal{P}'$ defined in Theorem 2.8 is well-behaved in the sense of Definition 2.1 (Honsell *et al.* 2013), and the predicate $\mathcal{P}'$ defined in Theorem 2.9 is well-behaved in the sense of Definition 2.1.

Proof. There is a logical chiasmus here. Theorem 2.8 is used to prove that the predicate $\mathcal{P}'$ in Theorem 2.9 is well-behaved, whereas Theorem 2.9 is used to prove that the predicate $\mathcal{P}'$ in Theorem 2.8 is well-behaved. □

3. The Logical Framework $\mathsf{CLLF}_{\mathcal{P}?}$

The main idea behind $\mathsf{CLLF}_{\mathcal{P}?}$ (see Figures 6, 7, and 8) is to "empower" the framework of $\mathsf{CLLF}_{\mathcal{P}}$ by *adding* to the lock/unlock mechanism the possibility to receive from the external oracle a *witness* satisfying suitable constraints. Thus, we can pave the way for plugging-in proof development environments beyond proof irrelevance scenarios. In this context, the lock constructor behaves as a *binding operator*. The new $(O{\cdot}Lock)$ rule is the following:

$$\frac{\Gamma, x{:}\sigma \vdash_{\Sigma} M \Leftarrow \rho}{\Gamma \vdash_{\Sigma} \mathcal{L}^{\mathcal{P}}_{x}[M] \Leftarrow \mathcal{L}^{\mathcal{P}}_{x,\sigma}[\rho]}$$

where the variable x is a placeholder bound in M and ρ, which will be replaced by the concrete term that will be returned by the external oracle call. The intuitive meaning behind the $(O{\cdot}Lock)$ 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 σ satisfying the constraint $\mathcal{P}$. The actual term, possibly synthesized by the external tool, will be "released" in $\mathsf{CLLF}_{\mathcal{P}?}$, by the unlock constructor in the $(O{\cdot}Unlock)$ rule as follows:

$$\frac{\Gamma \vdash_{\Sigma} A \Rightarrow \mathcal{L}^{\mathcal{P}}_{x,\sigma}[\rho] \quad \rho[N/x]^{\mathcal{F}}_{(\sigma)^-} = \rho' \quad \Gamma \vdash_{\Sigma} N \Leftarrow \sigma \quad \mathcal{P}(\Gamma \vdash_{\Sigma} N \Leftarrow \sigma)}{\Gamma \vdash_{\Sigma} \mathcal{U}^{\mathcal{P}}_{N}[A] \Rightarrow \rho'}$$

The term $\mathcal{U}^{\mathcal{P}}_{N}[A]$ intuitively means that N is precisely the synthesized term satisfying the constraint $\mathcal{P}(\Gamma \vdash_{\Sigma} N \Leftarrow \sigma)$ that will replace all the free occurrences of x in ρ. This replacement is executed in the $(\mathcal{S}{\cdot}O{\cdot}Unlock{\cdot}H)$ hereditary substitution rule (Figure 8).

Similarly to $\mathsf{CLLF}_{\mathcal{P}}$, it is possible also in $\mathsf{CLLF}_{\mathcal{P}?}$ to "postpone" or delay the verification

† Notice that the extra clause of rule $(O{\cdot}Nested{\cdot}Unlock)$, namely the occurrence of x in either $\mathcal{E}(M')$ or $\mathcal{E}(\rho')$, is granted by the fact that all judgments are in $\beta\eta\mathcal{UL}$-lnf and we are reasoning under the assumption that either M or ρ are not $\mathcal{U}^{\mathcal{P}}_{S,\sigma}[\,]$-free.

$$
\begin{array}{rcl@{\quad}rcl@{\quad}l}
\Sigma & \in & \mathcal{S} & \Sigma & ::= & \emptyset \mid \Sigma, a{:}K \mid \Sigma, c{:}\sigma & \textit{Signatures} \\
\Gamma & \in & \mathcal{C} & \Gamma & ::= & \emptyset \mid \Gamma, x{:}\sigma & \textit{Contexts} \\
K & \in & \mathcal{K} & K & ::= & \mathsf{type} \mid \Pi x{:}\sigma.K & \textit{Kinds} \\
\alpha & \in & \mathcal{A} & \alpha & ::= & a \mid \alpha\, N & \textit{Atomic Families} \\
\sigma, \tau, \rho & \in & \mathcal{F} & \sigma & ::= & \alpha \mid \Pi x{:}\sigma.\tau \mid \mathcal{L}^{\mathcal{P}}_{x,\sigma}[\rho] & \textit{Canonical Families} \\
A & \in & \mathcal{O}_a & A & ::= & c \mid x \mid A\, M \mid \mathcal{U}^{\mathcal{P}}_{N,\sigma}[A] & \textit{Atomic Objects} \\
M, N & \in & \mathcal{O} & M & ::= & A \mid \lambda x.M \mid \mathcal{L}^{\mathcal{P}}_{x}[M] & \textit{Canonical Objects}
\end{array}
$$

Fig. 6. $\mathsf{CLLF}_{\mathcal{P}?}$ Syntax

of an external predicate in a lock, provided an *outermost* lock is present. Whence, the synthesis of the actual inhabitant N can be delayed, thanks to the $(F{\cdot}Nested{\cdot}Unlock)$ and $(O{\cdot}Nested{\cdot}Unlock)$ rules, see Figure 7.

The Metatheory of $\mathsf{CLLF}_{\mathcal{P}?}$ follows closely that of $\mathsf{CLLF}_{\mathcal{P}}$ as far as decidability. We do not state a Correspondence Theorem since we did not introduce a non-canonical variant $\mathsf{CLLF}_{\mathcal{P}?}$. This could have been done similarly to $\mathsf{LLF}_{\mathcal{P}}$.

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 $\mathsf{CLLF}_{\mathcal{P}}$ and $\mathsf{CLLF}_{\mathcal{P}?}$. We discuss Fitch-Prawitz Consistent Set theory, FPST (Prawitz 1965), some applications of FPST to the normalizing λ-calculus, a system of normalizing call-by-value λ-calculus, a system of Light Linear Logic in $\mathsf{CLLF}_{\mathcal{P}}$, and an the encoding of *partial* functions in $\mathsf{CLLF}_{\mathcal{P}?}$.

The crucial step in encoding a logical system in $\mathsf{CLLF}_{\mathcal{P}}$ or $\mathsf{CLLF}_{\mathcal{P}?}$ is the definition of the predicates involved in locks in such a way that they are well-behaved. Predicates defined on closed terms are usually unproblematic. The difficulties arise from 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 Σ, let Λ_Σ (respectively, Λ^o_Σ) be the set of $\mathsf{CLLF}_{\mathcal{P}}$ terms (respectively, *closed* $\mathsf{CLLF}_{\mathcal{P}}$ terms) definable using constants from Σ. A term M has a *skeleton* in Λ_Σ 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 (Fitch 1952). This system was presented in Natural Deduction style by Prawitz (Prawitz 1965), with some modifications and improvements. Since Naïve Set Theory *à la* Cantor is inconsistent, Fitch's idea is to prevent the derivation of inconsis-

Signature rules

$$\frac{}{\emptyset \text{ sig}}\ (S{\cdot}Empty) \qquad \frac{\Sigma \text{ sig} \quad \vdash_\Sigma K \quad a \notin \mathsf{Dom}(\Sigma)}{\Sigma, a{:}K \text{ sig}}\ (S{\cdot}Kind) \qquad \frac{\Sigma \text{ sig} \quad \vdash_\Sigma \sigma \text{ type} \quad c \notin \mathsf{Dom}(\Sigma)}{\Sigma, c{:}\sigma \text{ sig}}\ (S{\cdot}Type)$$

Kind rules

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

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

Atomic Family rules

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

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

Canonical Family rules

$$\frac{\Gamma \vdash_\Sigma \alpha \Rightarrow \text{type}}{\Gamma \vdash_\Sigma \alpha \text{ type}}\ (F{\cdot}Atom)$$

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

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \rho \text{ type}}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{x,\sigma}[\rho] \text{ type}}\ (F{\cdot}Lock)$$

$$\frac{\Gamma, y : \tau \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{x,\sigma}[\rho] \text{ type} \quad \Gamma \vdash_\Sigma A \Rightarrow \mathcal{L}^{\mathcal{P}}_{x,\sigma}[\tau] \quad \rho[\mathcal{U}^{\mathcal{P}}_x[A]/y]^{F}_{(\tau)^-} = \rho' \quad x \in \mathsf{Fv}(\rho)}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{x,\sigma}[\rho']\text{type}}\ (F{\cdot}Nested{\cdot}Unlock)$$

Context rules

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

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

Atomic Object rules

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

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

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

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

Canonical Object rules

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

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

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

$$\frac{\Gamma, y{:}\tau \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_x[M] \Leftarrow \mathcal{L}^{\mathcal{P}}_{x,\sigma}[\rho] \quad \Gamma \vdash_\Sigma A \Rightarrow \mathcal{L}^{\mathcal{P}}_{x,\sigma}[\tau] \quad \rho[\mathcal{U}^{\mathcal{P}}_x[A]/y]^{F}_{(\tau)^-} = \rho' \quad M[\mathcal{U}^{\mathcal{P}}_x[A]/y]^{O}_{(\tau)^-} = M' \quad x \in \mathsf{Fv}(\rho) \text{ or } x \in \mathsf{Fv}(M)}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_x[M'] \Leftarrow \mathcal{L}^{\mathcal{P}}_{x,\sigma}[\rho']}\ (O{\cdot}Nested{\cdot}Unlock)$$

Fig. 7. The $\mathsf{CLLF}_{\mathcal{P}?}$ Type System

Substitution in Canonical Families

$$\frac{\sigma_1[M_0/x_0]^{\mathcal{F}}_{\rho_0} = \sigma'_1 \quad \sigma_2[M_0/x_0]^{\mathcal{F}}_{\rho_0} = \sigma'_2}{\mathcal{L}^{\mathcal{P}}_{x,\sigma_1}[\sigma_2][M_0/x_0]^{\mathcal{F}}_{\rho_0} = \mathcal{L}^{\mathcal{P}}_{x,\sigma'_1}[\sigma'_2]}\ (\mathcal{S}{\cdot}F{\cdot}Lock)$$

Substitution in Atomic Objects

$$(\mathcal{S}{\cdot}O{\cdot}Unlock{\cdot}H)$$
$$\frac{\sigma[M_0/x_0]^{\mathcal{F}}_{\rho_0} = \sigma' \quad M[M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = M' \quad M_1[M'/x]^{\mathcal{O}_a}_{(\sigma')^-} = M_2 \quad A[M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = \mathcal{L}^{\mathcal{P}}_x[M_1] : \mathcal{L}^{\mathcal{P}}_{x,\sigma'}[\rho]}{\mathcal{U}^{\mathcal{P}}_M[A][M_0/x_0]^{\mathcal{O}_a}_{\rho_0} = M_2 : \rho}$$

Substitution in Canonical Objects

$$\frac{\sigma_1[M_0/x_0]^{\mathcal{F}}_{\rho_0} = \sigma'_1 \quad M_1[M_0/x_0]^{\mathcal{O}}_{\rho_0} = M'_1}{\mathcal{L}^{\mathcal{P}}_x[M_1][M_0/x_0]^{\mathcal{O}}_{\rho_0} = \mathcal{L}^{\mathcal{P}}_x[M'_1]}\ (\mathcal{S}{\cdot}O{\cdot}Lock)$$

Fig. 8. $\mathsf{CLLF}_{\mathcal{P}?}$ Hereditary Substitution: changes w.r.t. $\mathsf{CLLF}_{\mathcal{P}}$

tencies from the unrestricted *abstraction* rule, by restricting the system FPST to allow for only normalizable *deductions*. In principle, this side-condition can be captured in LF, but only very deeply. Thus, the encoding becomes extremely cumbersome and obscure, and hence it violates the *de Brujin simplicity criterion*, making the proof of adequacy less "reliable". See (Honsell 2015) for more discussions on this point. An encoding in $\mathsf{CLLF}_{\mathcal{P}}$, on the other hand, factors out the machinery for enforcing the side-condition, breaking the task into a number of more elementary steps.

In this section we put to use the full machinery of $\mathsf{CLLF}_{\mathcal{P}}$ to provide an appropriate encoding of FPST where the *global* normalization constraint is enforced *locally* by checking the proof-object. This encoding illustrates beautifully the *bag of tricks* that $\mathsf{CLLF}_{\mathcal{P}}$ 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, could be freely replaced. We need to sterilize them, *i.e.* make them behave as axioms in the proof. To this end, we introduce a distinction between *generic* judgements, which cannot appear as conclusions of rule applications, but which can be *assumed* and *discharged*, 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. Once the distinction between judgments has been made, in Adequacy Theorems we consider only terms whose free judgement variables are of the generic type. No apodictic assumptions can be made at all.

Definition 4.2 (Fitch Prawitz Set Theory, FPST). For simplicity, here we only give the crucial rules for implication and for *set-abstraction* and the corresponding elimination

rules of the full system of Fitch (see (Prawitz 1965)), as presented by Prawitz:

$$\frac{\Gamma, A \vdash_{\mathsf{FPST}} B}{\Gamma \vdash_{\mathsf{FPST}} A \supset B}(\supset I) \qquad \frac{\Gamma \vdash_{\mathsf{FPST}} A \quad \Gamma \vdash_{\mathsf{FPST}} A \supset B}{\Gamma \vdash_{\mathsf{FPST}} B}(\supset E)$$

$$\frac{\Gamma \vdash_{\mathsf{FPST}} A[T/x]}{\Gamma \vdash_{\mathsf{FPST}} T \in \lambda x.A}(\lambda I) \qquad \frac{\Gamma \vdash_{\mathsf{FPST}} T \in \lambda x.A}{\Gamma \vdash_{\mathsf{FPST}} A[T/x]}(\lambda E)$$

The intended meaning of the term $\lambda x.A$ is the set $\{x \mid A\}$. 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 ($\mathsf{LLF}_{\mathcal{P}}$ signature Σ_{FPST} for Fitch Prawitz Set Theory). The following constants are introduced:

```
o   : Type                    ι : Type
T   : o -> Type               δ : Πa:o.(V(a) -> T(a))
V   : o -> Type          λ_intro : Πa:ι ->o.Πx:ι.T(a x) -> T(ϵ x (lam a))
lam : (ι -> o)-> ι        λ_elim : Πa:ι ->o.Πx:ι.T(ϵ x (lam a))->T(a x)
ϵ   : ι -> ι -> o        ⊃_intro : Πa,b:o.(V(a) -> T(b)) -> (T(a ⊃b))
⊃   : o -> o -> o ⊃_elim : Πa,b:o.Πx:T(a).Πy:T(a⊃b) -> L^Fitch_⟨x,y⟩,T(a)×T(a⊃b)[T(b)]
```

where o is the type of propositions, $\supset$ and the "membership" predicate ϵ are the syntactic constructors for propositions, lam is the "abstraction" operator for building "sets", T is the apodictic judgement, V is the generic judgement, δ is the coercion function, and $\langle \mathtt{x}, \mathtt{y} \rangle$ denotes the encoding of pairs, whose type is denoted by $\sigma \times \tau$, *e.g.* $\lambda \mathtt{u}{:}\sigma \to \tau \to \rho.\ \mathtt{u\ x\ y} : (\sigma \to \tau \to \rho) \to \rho$. The predicate in the lock is defined as follows:

$$\mathtt{Fitch}(\Gamma \vdash_{\Sigma_{\mathsf{FPST}}} \langle \mathtt{x}, \mathtt{y} \rangle \ \Leftarrow\ \mathtt{T(A)} \times \mathtt{T(A} \supset \mathtt{B)})$$

holds iff x and y have skeletons in $\Lambda_{\Sigma_{\mathsf{FPST}}}$, all the holes of which have either type o or are guarded by a δ, and hence have type V(A), and, moreover, the proof derived by combining the skeletons of x and y is normalizable in the natural sense. Clearly, this predicate is only semidecidable.

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(·), 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 o or are of generic judgement type. Adequacy for this signature can be achieved in the format of (Honsell *et al.* 2013):

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 $B_1 \ldots B_m \vdash_{\mathsf{FPST}} A$ iff there exists

a normalizable M such that $\mathtt{A_1}$:o, ..., $\mathtt{A_n}$:o, $\mathtt{x_1}$:V($\mathtt{B_1}$), ..., $\mathtt{x_m}$:V($\mathtt{B_m}$) $\vdash_{\Sigma_{\mathsf{FPST}}}$ M $\Leftarrow$ T(A) (where A, and $\mathtt{B_i}$ represent the encodings of, respectively, A and B_i in $\mathsf{CLLF}_{\mathcal{P}}$, for $1 \leq i \leq m$).

4.2. *A Type System for strongly normalizing λ-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 hence makes the system consistent. Of course, some intuitive rules are not derivable. For instance *modus ponens* does not hold and if $t \in \lambda x.A$ then we do not have necessarily that $A[t/x]$ holds. Similarly, the *transitivity* of implication does not hold. However FPST is a very expressive type system which "encompasses" many kinds of quantification, provided normalization is preserved, and Fitch has shown, see *e.g.* (Fitch 1952), that a large portion of ordinary Mathematics can be carried out in FPST.

In this subsection, we sketch how to use FPST to define a type system which can type *precisely all* the strongly normalizing λ-terms. Namely, we show that in FPST there exists a *syntax directed* inductive definition of a set Λ to which belong only the strongly normalizing λ-terms. We speak of a *type system* because the proof in FPST that a term belongs to Λ is *static*, in that it does not require to execute the term. Of course the normalization check is carried out at the metalevel when the quasi-deduction is deemed a deduction.

First, we need to be able to define recursive objects in FPST. We adapt, to FPST, Prop. 4, Appendix A.1 of (Girard 1998), originally given by J-Y. Girard for the Light Linear Logic version of Naïve Set Theory , 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 of the implication 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 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 $A[(\lambda w.\langle w, \Lambda\rangle \in \Lambda), x]$. □

Using the Fixpoint Theorem we define first natural numbers, then a concrete representation of the terms of λ-calculus, say Λ_0. Using again the Fixed Point Theorem, we define an (encoding of) the substitution function over terms in Λ_0 and finally the set Λ, such that $x \in \Lambda$ is equivalent in FPST to $x \in \Lambda_0 \& \forall y.y \in \Lambda \subset app(x, y) \in \Lambda$. Here, $app(x, y)$ denotes the concrete representation of "applying" x to y. One can derive in FPST that (a representation of) a λ-term, say M, belongs to Λ, only if there is a normalizable derivation of $M \in \Lambda$. But then it is straightforward to check that only *closed strongly normalizing terms* can be typed in FPST with Λ, *i.e.* belong to Λ. There is indeed a natural reflection of the metatheoretic 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 λ-calculus*

In order to illustrate further how to deal with free variables in defining well-behaved predicates, in this section we sketch how to express in $\mathsf{CLLF}_{\mathcal{P}}$ a call-by-value λ-calculus where β-reductions fire only if the operator is *normal* and delete arguments only if *normal*, namely

$$(\lambda x.M)N \to M[N/x] \quad \text{provided, } M \text{ normal and, if } x \notin M \text{ then } N \text{ normal.}$$

This calculus is *correct* w.r.t. the *observational semantics* defined as follows:

$$M =_{li} N \quad \equiv_{def} \quad \forall C[\,].C[M] \Downarrow_{li} \iff C[N] \Downarrow_{li}$$

where $P \Downarrow_{li}$ holds if the *leftmost innermost* reduction strategy terminates. In this theory, for example, the terms $(\lambda z.((\lambda x.\lambda y.y)(zz))(\lambda x.xx)$ and $(\lambda x.\lambda y.y)((\lambda x.xx)(\lambda x.xx))$ are not equated.

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

```
o      : Type       Eq  : o -> o -> Type       app : o -> o -> o
v      : Type       var : v -> o               lam : (v -> o) -> o
 refl : ΠM:o. (Eq M M)
 symm : ΠM,N:o. (Eq N M) -> (Eq M N)
trans : ΠM,N,P:o. (Eq M N) -> (Eq N P) -> (Eq M P)
eq_app : ΠM,N,M',N':o. (Eq M N) -> (Eq M'N') -> (Eq (app M M')(app N N'))
c_beta : ΠM:o->o,N:o.𝓛^{P^N}_{⟨M,N⟩,(o->o)×o}[Eq (app (lam λx:v.M(var x)) N) (M N)]
 csiv : ΠM,N:(v->o).(Πx:v.(Eq (M x)(N x)))->(Eq (lam M)(lam N))
```

where the predicate $\mathcal{P}^{\mathtt{N}}$ holds on $\Gamma \vdash_{\Sigma_{\lambda N}} \langle \mathtt{M}, \mathtt{N} \rangle \Leftarrow \mathtt{(o->o)\times o}$ if both `M` and `N` have skeletons in $\Lambda_{\Sigma_{\lambda N}}$ whose holes are guarded by a `var` (to prevent applications to non-normalizing terms of type `o`) and, moreover, M is "normal" and if $x \notin \mathtt{M}$ then `N` is "normal", in the intuitive sense outside terms guarded by a `var`.

Notice the role of `v`, which is akin to that of *generic* judgements in the FPST. Open terms are encoded by terms whose free variables are all of type `v`. All this is made explicit in the following adequacy result for this signature, which can be achieved only in the format of (Honsell *et al.* 2013), namely:

Theorem 4.3 (Adequacy for Normalizing call-by-value λ-calculus). if $v_1, \ldots, v_n$ are the variables occurring in $A_1,\ B_1,\ \ldots,\ A_m\ B_m,\ A\ B$, then

$$A_1 =_{\lambda_{li}} B_1,\ \ldots, A_m =_{\lambda_{li}} B_m \vdash_{\lambda N} A =_{\lambda_{li}} B$$

iff there exists `M` and $\mathtt{v_1}$:v, ..., $\mathtt{v_n}$:v such that

$$\mathtt{v_1}\text{:v},\ \ldots,\ \mathtt{v_n}\text{:v},\ \mathtt{x_1}\text{:Eq A}_1\ \mathtt{B}_1,\ \ldots,\ \mathtt{x_m}\text{:Eq A}_m\ \mathtt{B}_m \vdash_{\Sigma_{\lambda N}} \mathtt{M} \Leftarrow \mathtt{Eq\ A\ B}$$

4.4. *Elementary Affine Logic*

In this section we give a *shallow* encoding of *Elementary Affine Logic* as presented in (Baillot *et al.* 2007). 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 (Baillot *et al.* 2007)). Elementary Affine Logic can be specified by the following rules:

$$\frac{}{A \vdash_{EAL} A}\ (Var) \qquad \frac{\Gamma \vdash_{EAL} B}{\Gamma, A \vdash_{EAL} B}\ (Weak) \qquad \frac{\Gamma, A \vdash_{EAL} B}{\Gamma \vdash_{EAL} A \multimap B}\ (Abst) \qquad \frac{\Gamma \vdash_{EAL} A \quad \Delta \vdash_{EAL} A \multimap B}{\Gamma, \Delta \vdash_{EAL} B}\ (Appl)$$

$$\frac{\Gamma \vdash_{EAL} !A \quad \Delta, !A, \ldots, !A \vdash_{EAL} B}{\Gamma, \Delta \vdash_{EAL} B}\ (Contr) \qquad \frac{A_1, \ldots, A_n \vdash_{EAL} A \quad \Gamma_1 \vdash_{EAL} !A_1 \quad \ldots \quad \Gamma_n \vdash_{EAL} !A_n}{\Gamma_1 \ldots \Gamma_n \vdash_{EAL} !A}\ (Prom)$$

Definition 4.6 ($\mathsf{LLF}_{\mathcal{P}}$ signature Σ_{EAL} for Elementary Affine Logic).

```
o : Type   T : o -> Type   V : o -> Type   —o : o -> o -> o   ! : o -> o
c_appl   : ΠA,B :o. T(A) -> T(A —o B)-> T(B)  c_val : ΠA:o. V(A) -> T(!A)
c_abstr  : ΠA,B :o. Πx:(T(A) -> T(B)) -> L^Light_{x,T(A)->T(B)}[T(A —o B)]
c_promV_1 : ΠA,B :o. Πx:(T(A —o B)) -> L^Closed_{x,T(A—o B)}[T(!A) -> V(B)]
c_promV_2 : ΠA,B :o. Πx:(V(A —o B)) -> L^Closed_{x,V(A—o B)}[T(!A) -> V(B)]
```

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

— *Light* ($\Gamma \vdash_{\Sigma_{EAL}} \mathtt{x} \Leftarrow \mathtt{T(A)} \to \mathtt{T(B)}$) 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* ($\Gamma \vdash_{\Sigma_{EAL}} \mathtt{x} \Leftarrow \mathtt{T(A)}$) 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 (Baillot *et al.* 2007) is in effect a *family* of natural deduction rules with a growing number of assumptions. Our encoding achieves this via the auxiliary judgement `V`(·), whose effect is to allow for the user to mimick progressively the application of the promotion rule, as it is intuitively illustrated by the following steps (for the sake of readability and simplicity we drop the lock notation):

$\mathtt{T(A_1)->}\ldots\mathtt{->T(A_n)->T(A)}$ (first hypothesis of the promotion rule on paper)
$\mathtt{T(A_1)->T(A_2)->}\ldots\mathtt{->T(A_n} \multimap \mathtt{A)}$ (via an application of `c_abstr`)
$\vdots$
$\mathtt{T(A_1} \multimap (\mathtt{A_2} \multimap \ldots \multimap (\mathtt{A_n} \multimap \mathtt{A}) \ldots))$ (via n-applications of `c_abstr`)
$\mathtt{T(!A_1)->V(A_2} \multimap \ldots \multimap (\mathtt{A_n} \multimap \mathtt{A}) \ldots))$ (via an application of `c_promV_1`)
$\vdots$
$\mathtt{T(!A_1)->T(!A_2)->}\ldots\mathtt{->T(!A_n)->V(A)}$ (via n-applications of `c_promV_2`)
$\mathtt{T(!A_1)->T(!A_2)->}\ldots\mathtt{->T(!A_n)->T(!A)}$ (via an application of `c_val`)

Hence, starting from the hypothesis that $\mathtt{T(A)}$ follows from $\mathtt{T(A_1)},\ldots,\mathtt{T(A_n)}$ (first hypothesis of the promotion rule "on paper") and knowing that $\mathtt{T(!A_1)},\ldots,\mathtt{T(!A_n)}$ also hold (the

remaining hypotheses of the promotion rule), we can infer that $\mathtt{T(!A)}$ holds (conclusion of the promotion rule). Notice that only after the last step we have fully represented the intended semantics of the promotion rule. Of course at each step where `c_abstr`, `c_promV_1`, and `c_promV_2` are applied one must verify the constraints imposed by the corresponding locks (not showed above in order to avoid hindering the readability of the derivation sketch).

Adequacy for this signature can be achieved only in the format of (Honsell *et al.* 2013), namely:

Theorem 4.4 (Adequacy for Elementary Affine Logic). if $A_1, \ldots, A_n$ are the atomic formulas occurring in $B_1, \ldots, B_m, A$, then $B_1 \ldots B_m \vdash_{EAL} A$ iff there exists $\mathtt{M}$ and $\mathtt{A_1{:}o}, \ldots, \mathtt{A_n{:}o}, \mathtt{x_1{:}\,T(B_1)}, \ldots, \mathtt{x_m{:}T(B_m)} \vdash_{\Sigma_{EAL}} \mathtt{M} \Leftarrow \mathtt{T(A)}$ (where $\mathtt{A}$, and $\mathtt{B_i}$ represent the encodings of, respectively, A and B_i in $\mathsf{CLLF}_{\mathcal{P}}$, for $1 \leq i \leq m$) and all variables $\mathtt{x_1} \ldots \mathtt{x_m}$ occurring more than once in $\mathtt{M}$ have type of the shape $\mathtt{T(B_i)} \equiv \mathtt{T(!C_i)}$ for some suitable formula $\mathtt{C}_i$.

The check on the context of the Adequacy Theorem is *external* to the system $\mathsf{LLF}_{\mathcal{P}}$, but this is in the nature of results which relate *internal* and *external* concepts. For example, the very concept of $\mathsf{LLF}_{\mathcal{P}}$ context, which appears in any adequacy result, is external to $\mathsf{LLF}_{\mathcal{P}}$. Of course, this check is internalized if the term is closed.

4.5. *Square roots of natural numbers in* $\mathsf{CLLF}_{\mathcal{P}?}$

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 $\mathsf{CLLF}_{\mathcal{P}?}$ 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 $\mathsf{CLLF}_{\mathcal{P}?}$.

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

```
nat: type  0: nat  S: nat->nat  plus : nat->nat->nat  minus : nat->nat->nat
mult  : nat->nat->nat      sqroot: nat->nat           eval : nat->nat->type
sqrt  : Πx:nat.L^SQRT_{y,nat×σ}[(eval (sqroot x) (fst y))]
```

Thus, we have the usual representation of natural numbers by means of a constant `0` representing zero and the symbol `S` representing the successor. Moreover, we have the classical operations + and * which are represented by `plus` and `mult`, while $\dot{-}$ (represented by `minus` in our signature) is defined as follows:

$$\mathtt{x} \dot{-} \mathtt{y} \triangleq \begin{cases} \mathtt{x - y} & \mathtt{if\ x \geq y} \\ \mathtt{0} & \mathtt{otherwise} \end{cases}$$

Finally, `eval` represents the usual evaluation predicate relating arithmetical expressions on natural numbers to the corresponding results (*e.g.*, the derivation of a term with type `(eval (mult (S (S 0)) (S (S 0))) (S (S (S (S 0)))))` represents the evaluation of the arithmetic expression $2 * 2$ to the corresponding result 4). For the sake of brevity,

we do not give here the complete set of rules involving the `eval` predicate, but we focus on the definition of the type of the `sqrt` constant (representing the square root case), where the variable `y` is a pair, the type σ is defined as follows:

(`eval` (`plus` (`minus` `x` (`mult` (`fst` `y`) (`fst` `y`))) (`minus` (`mult` (`fst` `y`) (`fst` `y`)) `x`) `0`)),

namely the judgement which encodes $(\mathtt{x} \dot{-} \mathtt{N} * \mathtt{N}) + (\mathtt{N} * \mathtt{N} \dot{-} \mathtt{x}) = \mathtt{0}$, and the external predicate $SQRT(\Gamma \vdash_\Sigma \mathtt{y} \Leftarrow \mathtt{nat} \times \sigma)$ is defined to hold if and only if `(fst y)` (*i.e.*, the first projection of `y`) is the minimum natural number `N` such that $(\mathtt{x} \dot{-} \mathtt{N} * \mathtt{N}) + (\mathtt{N} * \mathtt{N} \dot{-} \mathtt{x}) = \mathtt{0}$. This is precisely the definition of the square root in the domain of natural numbers.

Notice that the user does not have to provide a "deep" specification of the computation algorithm of square roots in the framework $\mathsf{CLLF}_{\mathcal{P}?}$. Indeed, its semantics is implicit in the definition of the predicate *SQRT*, whence it relies entirely on the external oracle which can be a specialized software being able to perform the computation of the square root very efficiently.

5. Related work and Future Perspectives

Building a universal proof metalanguage where different tools and formalisms can be "plugged in" and "glued together" is a long standing goal that has been extensively explored in a vast and inspiring literature on Logical Frameworks by (Barthe *et al.* 2003; Pfenning and Schürmann 1999; Watkins *et al.* 2002; Schack-Nielsen and Schürmann 2008; Cousineau and Dowek 2007; Boespflug *et al.* 2012; Nanevski *et al.* 2008; Pientka *et al.* 2008; Pientka *et al.* 2010; Honsell *et al.* 2007; Honsell *et al.* 2012; Honsell 2013; Wang and Chaudhuri 2015; Battel and Felty 2015). The clear-cutting monadic structure and properties of the lock/unlock mechanism go back to Moggi's notion of computational monads (Moggi 1989) and our system can be seen as a generalization, to a family of dependent *lax* operators, of Moggi's *partial* λ-calculus (Moggi 1988) and of the work carried out in (Fairtlough and Mendler 1997; Mendler 1991) (which is also the original source of the term "lax"). A correspondence between lax modalities and monads in functional programming was pointed out in (Alechina *et al.* 2001; Garg and Tschantz 2008). On the other hand, although the connection between constraints and monads in logic programming was considered in the past, *e.g.*, in (Nanevski *et al.* 2008; Fairtlough *et al.* 1997; Fairtlough and Mendler 2001), 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 capitalizing on this in logical frameworks. Of course, there are a lot of interesting points of contact with other systems in the literature which should be explored further. For instance, in (Nanevski *et al.* 2008), 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 `let` construct in favor of a deeper substitution. Schröder-Heister has discussed in a number of papers, see *e.g.* (Schroeder-Heister 2012b; Schroeder-Heister 2012a), 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, which should be explored. 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 and the distinction between *generic* and *apodictic* judgements. The system LFSC, presented in (Stump 2008; Stump *et al.* 2012), 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 factors the verifications of side-conditions out of the main proof. The 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 SMT solvers.

Practical implementations of $\mathsf{CLLF}_{\mathcal{P}}$ and especially $\mathsf{CLLF}_{\mathcal{P}?}$ need to be experimented with. Moreover, it would be important to develop the possibility of introducing and capitalizing on a *logical algebra* of well-behaved predicates. For instance, it would be interesting to compare $\mathsf{CLLF}_{\mathcal{P}?}$ with the system Why3 (Filliâtre 2013), in the field of program verification based on Hoare's Logic.

References

N. Alechina, M. Mendler, V. De Paiva, E. Ritter. Categorical and Kripke semantics for constructive s4 modal logic. In *Computer Science Logic*, pp. 292–307. Springer, 2001, doi:10.1007/3-540-44802-0_21.

P. Baillot, P. Coppola, U. Dal Lago. Light logics and optimal reduction: Completeness and complexity. In *LICS*, pp. 421–430. IEEE Computer Society, 2007, doi:10.1016/j.ic.2010.10.002.

H.P. Barendregt, E. Barendsen. Autarkic computations in formal proofs. *Journal of Automated Reasoning*, 28:321–336, 2002, doi:10.1.1.39.3551.

G. Barthe, H. Cirstea, C. Kirchner, L. Liquori. Pure Pattern Type Systems. In *POPL'03*, pp. 250–261, ACM, doi:10.1.1.298.4555.

C. Battel, A. Felty. A Higher-Order Logical Framework for Reasoning about Programming Languages. In Proc. of CMS Winter Meeting, Montréal, December 4–7, 2015.

M. Boespflug, Q. Carbonneaux, O. Hermant. The λΠ-calculus modulo as a universal proof language. In *PxTP 2012*, v. 878, pp.28–43, 2012, doi:10.1.1.416.1602.

R. Boulton, A. Gordon, M. Gordon, J. Harrison, J. Herbert, J. Van Tassel. Experience with embedding hardware description languages in HOL. In *TPCD*, pp. 129–156. North-Holland, 1992, doi:10.1.1.111.260.

D. Cousineau, G. Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In *TLCA*, v. 4583 of *LNCS*, pp. 102–117. Springer-Verlag, 2007, doi:10.1.1.102.4096.

M. Fairtlough, M. Mendler. Propositional lax logic. *Information and Computation*, 137(1):1–33, 1997, doi:10.1.1.22.5812.

M. Fairtlough, M. Mendler, X. Cheng. Abstraction and refinement in higher order logic. In *Theorem Proving in Higher Order Logics*, pp. 201–216. Springer, 2001, doi:10.1.1.29.3515.

M Fairtlough, M. Mendler, M. Walton. First-order lax logic as a framework for constraint logic programming. Technical report, 1997, doi:10.1.1.36.1549.

J.-C. Filliâtre. One Logic To Use Them All. In Proc. of CADE 24 - the 24th International Conference on Automated Deduction, Lake Placid, NY, United States, editor: Maria Paola Bonacina, Jun 2013.

F. B. Fitch. *Symbolic logic - An Introduction*. New York, 1952, ASIN: B0007DLS2O.

D. Garg, M. C. Tschantz. From indexed lax logic to intuitionistic logic. Tech. rep. CMU, 2008, doi:10.1.1.295.8643.

J.-Y. Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998, doi:10.1.1.134.4420.

R. Harper, D. Licata. Mechanizing metatheory in a logical framework. *JFP*, 17:613–673, 2007, doi:10.1017/S0956796807006430.

D. Hirschkoff. Bisimulation proofs for the π-calculus in the Calculus of Constructions. In *TPHOL'97*, n. 1275 in LNCS. Springer, 1997, doi:10.1007/BFb0028392.

F. Honsell, M. Miculan, I. Scagnetto. π-calculus in (Co)Inductive Type Theories. *Theoretical Computer Science*, 253(2):239–285, 2001, doi:10.1016/S0304-3975(00)00095-5.

F. Honsell, M. Lenisa, L. Liquori. A Framework for Defining Logical Frameworks. *Volume in Honor of G. Plotkin, ENTCS*, 172:399–436, 2007, doi:10.1016/j.entcs.2007.02.014.

F. Honsell, M. Lenisa, L. Liquori, P. Maksimovic, I. Scagnetto. $\mathsf{LF}_{\mathcal{P}}$: a logical framework with external predicates. In *LFMTP*, pp. 13–22. ACM, 2012, doi:10.1145/2364406.2364409.

F. Honsell, M. Lenisa, L. Liquori, P. Maksimovic, I. Scagnetto. An open logical framework. Accepted for publication in *Journal of Logic and Computation*, 2013, doi:10.1093/logcom/ext028.

F. Honsell. 25 years of formal proof cultures: Some problems, some philosophy, bright future. In *LFMTP'13*, pp. 37–42, ACM, 2013, doi:10.1145/2503887.2503896.

F. Honsell, L. Liquori, I. Scagnetto. $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$: Side Conditions and External Evidence as Monads. In *MFCS 2014, Part I*, v. 8634 of *LNCS*, pp. 327–339, Budapest, Hungary, August 2014. Springer, doi:10.1007/978-3-662-44522-8_28.

F. Honsell, L. Liquori, P. Maksimovic, I. Scagnetto. Gluing together Proof Environments: Canonical extensions of LF Type Theories featuring Locks. In Proc. of LFMTP 2015, Berlin, Germany, 01/08/2015, pp. 3–17, http://dx.doi.org/10.4204/EPTCS.185.1, ISSN: 2075-2180, Open Publishing Association.

F. Honsell. Wherefore thou art ... Semantics of Computation? In Proc. of HaPoC 2015 : 3rd International CONFERENCE on the HISTORY and PHILOSOPHY of COMPUTING, 8–11 Oct 2015 Pisa (Italy), to appear.

F. Honsell, L. Liquori, P. Maksimovic, I. Scagnetto. $\mathsf{LLF}_{\mathcal{P}}$: A Logical Framework for modeling External Evidence, Side Conditions, and Proof Irrelevance using Monads. Accepted for publication in the Special Issue of Logical Methods in Computer Science devoted to Festschrift for Pierre-Louis Curien, preliminary version available at http://www.dimi.uniud.it/scagnett/LLFP_LMCS.pdf, 2016.

M. Kerber. A Dynamic Poincaré Principle. In Proc. of Mathematical Knowledge Management - 5th International Conference, MKM 2006; editor: Jonathan M. Borwein and William M. Farmer, pages 44–53, Springer, LNAI 4108, 2006

M. Mendler. Constrained proofs: A logic for dealing with behavioral constraints in formal hardware verification. In *Designing Correct Circuits*, pp. 1–28. Springer-Verlag, 1991, doi:10.1007/978-1-4471-3544-9_1.

E. Moggi. *The partial lambda calculus*. PhD thesis, University of Edinburgh, 1988, doi:10.1.1.53.8462.

E. Moggi. Computational lambda-calculus and monads. In *LICS 1989*, pp. 14–23. IEEE Press, doi:10.1.1.26.2787.

A. Nanevski, F. Pfenning, B. Pientka. Contextual Modal Type Theory. *ACM TOCL*, 9(3), 2008, doi:10.1145/1352582.1352591.

F. Pfenning, C. Schürmann. System description: Twelf – a meta-logical framework for deductive systems. In *CADE*, v. 1632 of *LNCS*, pp. 202–206. Springer-Verlag, 1999, doi:10.1007/3-540-48660-7_14.

B. Pientka, J. Dunfield. Programming with proofs and explicit contexts. In *PPDP'08*, pp. 163–173, ACM, doi:10.1145/1389449.1389469.

B. Pientka, J. Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In *IJCAR 2010*, v. 6173 of *LNCS*, pp. 15–21. Springer-Verlag, doi:10.1007/978-3-642-14203-1_2.

D. Prawitz. *Natural Deduction. A Proof Theoretical Study.* Almqvist Wiksell, Stockholm, 1965, ISBN: 978-0486446554.

A. Schack-Nielsen, C. Schürmann. Celf–A logical framework for deductive and concurrent systems (System description). in *Automated Reasoning*, pp. 320–326, Springer, 2008, doi:10.1007/978-3-540-71070-7_28.

P. Schroeder-Heister. Paradoxes and Structural Rules. *Insolubles and consequences : essays in honor of Stephen Read*, pp. 203–211. College Publications, London, 2012, ISBN 978-1-84890-086-8.

P. Schroeder-Heister. Proof-theoretic semantics, self-contradiction, and the format of deductive reasoning. *Topoi*, 31(1):77–85, 2012, doi:10.1007/s11245-012-9119-x.

A. Stump. Proof checking technology for satisfiability modulo theories. In *LFMTP 2008*, v. 228, pp. 121–133, 2009, doi:10.1.1.219.1459.

A. Stump, A. Reynolds, C. Tinelli, A. Laugesen, H. Eades, C. Oliver, R. Zhang. LFSC for SMT Proofs: Work in Progress. In Proceedings of the 2nd International Workshop on Proof eXchange for Theorem Proving (PxTP'12), Manchester, UK, 2012.

Y. Wang, K. Chaudhuri. A Proof-theoretic Characterization of Independence in Type Theory. 13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015), vol. 38, pages 332–346, 2015

K. Watkins, I. Cervesato, F. Pfenning, D. Walker. A Concurrent Logical Framework I: Judgments and Properties. Tech. Rep. CMU-CS-02-101, CMU, 2002, doi:10.1.1.14.5484.