25 Years of Formal Proof Cultures: Some Problems, Some Philosophy, Bright Future

Furio Honsell

Extended Abstract: 25 Years of Formal Proof Cultures

some problems, some philosophy, bright future

Furio Honsell
Università di Udine, Italy
{surname.name}@uniud.it

Abstract

Throughout the history of Mathematics, several different *proof cultures* have co-existed, and still do co-exist. After 25 years of Logical Frameworks, we can say that even as far as *proof metalanguages* go, a definitive system is utopian and that we are witnessing the continuous development of a diversity of *formal* proof cultures, see *e.g.* [10–12, 17, 19, 21, 23, 24, 28]. In this paper, we propose a contribution towards the clarification of some controversial issues that have arisen in the theory and practice of Logical Frameworks, and have possibly motivated such a manifold speciation. Using as a running example the encoding of the critical features of Non-Commutative Linear Logic (NCLL) [26] in the Logical Framework $\mathsf{LF}_{\mathcal{P}}$ [20], we discuss the notions of *adequacy* of an encoding, *locality* of a side-condition, *deep and shallow* encodings, and how to embed *heterogenous* justifications or *external evidence* in LF. This discussion naturally leads to the question of how to express formally the *expressive power* of a Logical Framework, a minimal requirement being that of encoding *itself within itself*. We focus on $\mathsf{LF}_{\mathcal{P}}$ and we discuss its relations to the original LF [17], and briefly to the Conditional LF [21], and the Pattern LF [19] previously introduced by the authors. We conclude the paper by briefly comparing $\mathsf{LF}_{\mathcal{P}}$ to $\lambda\Pi$-calculus modulo [12], the Linear LF [9], and the Concurrent LF[28].

Categories and Subject Descriptors F.3.1 [*Specifying and Verifying and Reasoning about Programs*]: Mechanical verification

General Terms Theory, Verification

Keywords Type Theory, Logical Frameworks

1. Introduction

Logical Frameworks (LFs) first came into being in 1987 [17], as an attempt to harness the multifarious variety of existing logics and formal proof cultures, with the aim of factoring out the complexities of implementing proof assistants for such systems, especially *program logics*. LFs are meta-languages for specifying the *assertive* and *deductive machinery* of logical systems and, most notably, what should count as formal *justifications*, *evidence* or *proofs*, in deductive systems.

However, the very concept of *proof*, let alone that of *formal* proof, has remained quite elusive throughout the history of Mathematics. In ancient Egyptian, Mesopotamic, and Indian Mathematics justification often reduced just to *demonstrations or diagrams*. Euclid, in hellenistic times, was the first to set a standard for an apparently precise notion of formal proof, but he himself was rather easy in the Elements in using diagrams (*e.g.* continuity) and proving universal statements only for specific instances (*e.g.* the existence of infinitely many primes). Archimedes often used *physical analogies*. And even nowadays, many working mathematicians have an ambivalent attitude towards formal proof, considering them a pedantry, while complaining if computers provide formal dependability. And even today, when it is acknowledged that FOL provides, in principle, the ultimate logical answer, the multitude of irreducible logics is increasing.

As there was no hope for an *ultimate logical system*, 25 years ago a utopia was envisaged where at least an ultimate *metalanguage* for logics could exist. Today, we think that even this very modest universality is probably unattainable. A number of Logical Frameworks has been put forward in recent decades [11, 12, 17, 19, 21, 23, 24, 28] each convincingly motivating the peculiar features of the formal proof culture it supports. Albeit, in principle, they are not *incommensurable*, this is not the case pragmatically.

The main problem in devising a Logical Framework is that of deciding *"which features of the object systems should be delegated in a transparent way to the metalanguage"*. The difficulty is that usually many pragmatical features of object systems, including psychological ones, are not *invariant* under encodings.

After 25 years, we think that a minimal answer is provided by Logical Frameworks, inspired by LF, based on *Intuitionistic Dependent Type Theories*. These rely upon the methodology of *Higher Order Abstract Syntax* (HOAS), for encoding logical languages, and the paradigms of *theorems-as-types, proofs-as-terms*, and *rules-as-functions*. Variables, rule instantiations, substitution, assumptions, as well as their management according to a stack-discipline, are taken care of straightforwardly by the framework. Often, however, this approach does not encompass all *side conditions*, thus leading to rather *deep* encodings and the introduction of a huge spectrum of extra judgements which obscure the very point for having introduced the object system in the first place.

In our view, there are two main issues, which are nonetheless somehow related. The first is: should we enrich the LF with more specific structural operations as in *e.g.* CLF [28] which can enforce linearity and can deal directly with patterns? The second issue is: how should we combine deduction with heterogenous sources of justification, *e.g. computation* as in [10] and in $\lambda\Pi$-calculus *modulo*. The first issue has a bearing on the depth of the encoding. The more shallow the encoding, the more transparent the Logical Framework will be for the user. The second issue has a bearing on the reliability and efficiency of the proof checker, *i.e.* the so-called *de Bruijn criterion* [14].

Σ	$\in$	*Signatures*	Σ	$::=$	$\emptyset \mid \Sigma, a{:}K \mid \Sigma, c{:}\sigma$
Γ	$\in$	*Contexts*	Γ	$::=$	$\emptyset \mid \Gamma, x{:}\sigma$
K	$\in$	*Kinds*	K	$::=$	$\mathsf{Type} \mid \Pi x{:}\sigma.K$
σ, τ, ρ	$\in$	*Families (Types)*	σ	$::=$	$a \mid \Pi x{:}\sigma.\tau \mid \sigma\ N \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$
M, N	$\in$	*Objects*	M	$::=$	$c \mid x \mid \lambda x{:}\sigma.M \mid M\ N \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \mid \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]$

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

In recent years, the authors have been pursuing a Logical Framework that would provide a setting where these issues could at least be openly expressed. First, we introduced the Schematic Logical Framework [19], and discussed the Pattern Logical Framework PLF as an instance. Then we reduced this generality by introducing first the Conditional Logical Framework [21], and finally $\mathsf{LF}_{\mathcal{P}}$ [20], which is an *Open* Logical Framework. We think that this latter framework is *potentially* very expressive, though retaining modesty, whereby other frameworks can be compared.

In this paper, we offer a contribution towards the clarification of some controversial issues that have arisen in the past 25 years of theory and practice of Logical Frameworks and have possibly motivated their manifold speciation. Using as a running example the encoding of the critical features of Non-Commutative Linear Logic (NCLL) [26] in the Logical Framework $\mathsf{LF}_{\mathcal{P}}$ [20], we discuss the notions of *adequacy* of an encoding, *locality* of a side-condition, *deep and shallow* encodings, and how to embed *heterogenous* justifications in LF. This discussion naturally takes us to the question of how to discuss formally the *expressive power* of a Logical Framework, a minimal standard being that of encoding *itself within itself*. We focus on $\mathsf{LF}_{\mathcal{P}}$ and we discuss its relations to the original LF [17], and briefly to the Conditional LF [21] and the Pattern LF [19]. We conclude the paper by briefly extending the comparison to the $\lambda\Pi$-calculus modulo [12], LLF [9], and CLF [28].

2. $\mathsf{LF}_{\mathcal{P}}$

The system $\mathsf{LF}_{\mathcal{P}}$ [20] is a decidable conservative extension of LF. It was introduced to neatly factor out computations whose justifications could be delegated to an *external oracle*. In our view, this allows us to recover within the Logical Framework many different proof cultures that could previously be embedded only very deeply and axiomatically. Recourse in formal proofs to diverse *non-apodictic* sources of justifications and *external evidence*, such as diagrams, physical analogies, explicit computation according to the *Poincaré Principle* [3], external proof search tools, can thus be explicitly *recorded* in a LF type-theoretic framework. This is not too superficial, since the *execution* of just about any proof procedure requires some irreducible assumption, as illustrated by the *Münchausen trilemma* or the story of *Achilles and the tortoise* as narrated by Lewis Carroll [8].

In this section, we briefly recall the syntax and the basic notions underpinning $\mathsf{LF}_{\mathcal{P}}$: In Figure 1, we give the syntactic categories of $\mathsf{LF}_{\mathcal{P}}$, namely signatures, contexts, kinds, families (*i.e.*, types) and objects (*i.e.*, terms).

The novelties of $\mathsf{LF}_{\mathcal{P}}$ w.r.t. classic LF are the lock ($\mathcal{L}$) and unlock ($\mathcal{U}$) operators. Indeed, $\mathsf{LF}_{\mathcal{P}}$ is an extension of LF with *external predicates* and, precisely in this sense it is an *Open* Logical Framework. The *lock type constructors* act as *logical filters* and introduce a sort of $\Diamond\Box$-*modality constructors* for building types of the shape $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, where $\mathcal{P}$ is a predicate on typed judgements.

Following the standard specification paradigm in Constructive Type Theory, we define lock-types using *introduction*, *elimination*, and *equality rules*. Namely, we introduce a lock-*constructor* for building objects $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]$ of type $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, via the *introduction rule* $(O{\cdot}Lock)$. Correspondingly, we introduce an unlock-*destructor*, $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]$, and an *elimination rule* $(O{\cdot}Unlock)$, which allows for the elimination of the lock-type constructor, under the condition that a specific predicate $\mathcal{P}$ is verified, possibly *externally*, on an appropriate *correct*, *i.e.* derivable, judgement.

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

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

The *equality rule* for lock-types amounts to a lock-reduction ($\mathcal{L}$-reduction), $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\mathcal{L}} M$, which allows for the elimination of a *lock*, in the presence of an *unlock*. The $\mathcal{L}$-reduction combines with standard β-reduction into $\beta\mathcal{L}$-reduction. But, since external predicates affect reductions in $\mathsf{LF}_{\mathcal{P}}$, they must be *well behaved* in order to maintain Subject Reduction, and hence *decidability* as all LF *must do*.

DEFINITION 1 (Well-behaved predicates). *A finite set of predicates $\{\mathcal{P}_i\}_{i\in I}$ is* well-behaved *if each $\mathcal{P}$ in the set satisfies the following conditions:*

- ***Closure under signature and context weakening and permutation:***
  1. *If Σ and Ω are valid signatures such that $\Sigma \subseteq \Omega$, and $\mathcal{P}(\Gamma \vdash_\Sigma \alpha)$ holds, then $\mathcal{P}(\Gamma \vdash_\Omega \alpha)$ also holds.*
  2. *If Γ and Δ are valid contexts such that $\Gamma \subseteq \Delta$, and $\mathcal{P}(\Gamma \vdash_\Sigma \alpha)$ holds, then $\mathcal{P}(\Delta \vdash_\Sigma \alpha)$ also holds.*
- ***Closure under substitution:***
  If $\mathcal{P}(\Gamma, x{:}\sigma', \Gamma' \vdash_\Sigma N : \sigma)$ holds, and $\Gamma \vdash_\Sigma N' : \sigma'$, then $\mathcal{P}(\Gamma, \Gamma'[N'/x] \vdash_\Sigma N[N'/x] : \sigma[N'/x])$ also holds.
- ***Closure under reduction:***
  1. *If $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$ holds, and $N \to_{\beta\mathcal{L}} N'$ holds, then $\mathcal{P}(\Gamma \vdash_\Sigma N' : \sigma)$ also holds.*
  2. *If $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$ holds, and $\sigma \to_{\beta\mathcal{L}} \sigma'$ holds, then $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma')$ also holds.*

In the following subsection we use the notions of *adequacy* and *depth* of an encoding, which will be discussed in Section 4.

2.1 $\mathsf{LF}_{\mathcal{P}}$ in $\mathsf{LF}_{\mathcal{P}}$

A minimal *expressivity* requirement for a Logical Framework is to be able to represent itself, see *e.g.* [1, 5].

In encoding $\mathsf{LF}_{\mathcal{P}}$ within $\mathsf{LF}_{\mathcal{P}}$ we start by introducing four types to represent kinds, types, terms, kinds, and formulæ:

```
kind: Type,      tp:Type,      term: Type,      o:Type
```

Then, we introduce term and type constructors[1]:

```
type :  kind
prodk:  (term -> kind) -> kind

prodt:  (term -> tp)   -> tp
tp_app:    tp -> term  -> tp
tp_Lock:(term -> o)    -> tp   -> term -> tp -> tp

app:     term -> term  -> term
abs:    (term -> term) -> term
Lock:   (term -> o)    -> term -> term -> tp -> term
Unlock: (term -> o)    -> term -> term -> tp -> term
```

The adequacy of the signature given so far (and denoted by Σ) is straightforward.

Capitalizing on the multiple uses of metavariables, we do not need to represent explicitly typing environments and signatures. It is sufficient to "record" the typings by means of a *bookkeeping* judgment:

[1] Following the spirit of the Edinburgh LF, we identify the variables of the object language with metavariables of the suitable type.

```
tp_typing: tp   -> kind -> Type
typing   : term -> tp   -> Type
```

We give the encodings of only two typing rules, namely, $(O{\cdot}Lock)$ and $(O{\cdot}Unlock)$ and of the Lock-reduction rule:

```
OLock:   ΠM,N:term. Πrho,sigma:tp. ΠP:term -> o.
  (typing M rho) -> (typing N sigma) ->
  (typing (Lock P M N sigma) (tp_Lock P rho N sigma))

OUnlock: ΠM,N:term. Πrho,sigma:tp. ΠP:term -> o.
  (typing M (tp_Lock P rho N sigma)) -> (typing N sigma) ->
  L^isTrue_⟨P,N⟩[(typing (Unlock P M N sigma) rho)]

UL:      ΠM,N:term. ΠP:term -> o. Πsigma:tp.
  L^isTrue_⟨P,N⟩[(red (Unlock P (Lock P M N sigma) N sigma) M)]
```

where `red: term -> term -> Type` denotes the $\beta\mathcal{L}$-reduction judgment on terms and the *external* predicate `isTrue` holds on the pair $\langle$`P`,`N`$\rangle$ iff predicate `P` holds on `N`.

The encoding of the typing judgment on terms and types, namely `typing`, is adequate in the usual sense given by the following theorem, where $\epsilon_{\mathcal{X}}$ stands for the encoding function mapping terms and types of $\mathsf{LF}_{\mathcal{P}}$ with free variables in $\mathcal{X}$ into the corresponding canonical forms of type `term` and `tp`, respectively:

THEOREM 1 (Adequacy of typing). *Given* $\mathcal{X} = \{x_1, \ldots, x_n\}$ *be the set of free variables occurring in* M *and* σ *and* $\Gamma = [x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n]$*, then there is a bijection between derivations of the judgment* $\Gamma \vdash M : \sigma$ *in* $\mathsf{LF}_{\mathcal{P}}$ *and proof terms* `h`*, such that* $\Gamma' \vdash_{\Sigma}$ `h` $:$ (`typing` $\epsilon_{\mathcal{X}}(M)$ $\epsilon_{\mathcal{X}}(\sigma)$) *is in canonical form (where* $\Gamma' = \{$`x1` $:$ `term`, $\ldots$, `xn` $:$ `term`, `h1` $:$ (`typing x1` $\epsilon_{\mathcal{X}}(\sigma_1)$), $\ldots$, `hn` $:$ (`typing xn` $\epsilon_{\mathcal{X}}(\sigma_n)$)$\}$*).* ■

Similar statements can be proved for judgments `red` and `tp_typing`.

2.2 Comparing $\mathsf{LF}_{\mathcal{P}}$ and LF

We have the following result:

THEOREM 2. $\mathsf{LF}_{\mathcal{P}}$ *is a conservative extension of* LF. ■

Proof (sketch): consider a derivation in $\mathsf{LF}_{\mathcal{P}}$ and drop all occurrences of locks and unlocks (that is, *release* the terms and types originally locked). So doing, we obtain a legal derivation in standard LF, since the terms and types encapsulated by the $\mathcal{L}$ and $\mathcal{U}$-constructors are "fully compatible" with the typing system of LF.

Notice that $\mathsf{LF}_{\mathcal{P}}$ is a conservative extension of LF independently of the particular nature and properties of the external oracles we may *invoke* during the proof development (in $\mathsf{LF}_{\mathcal{P}}$), *e.g.* decidability or polynomial complexity of $\mathcal{P}$. But, if we consider *well-behaved recursively enumerable predicates*, then these are definable in LF by Church's thesis. Thus, we can envisage a *deep* embedding of $\mathsf{LF}_{\mathcal{P}}$ into LF, in the style of Section 2.1, which internalizes also the external predicates and the decision procedures related to the lock/unlock mechanism and $\mathcal{L}$-reduction as follows:

```
OUnlock: ΠM,N:term. Πrho,sigma:tp. ΠP:term -> o.
 (typing M (tp_Lock P rho N sigma)) -> (typing N sigma) ->
 (isTrue P N) -> (typing (Unlock P M N sigma) rho)

UL:      ΠM,N:term. ΠP:term -> o. Πsigma:tp.
 (isTrue P N) ->
 (red (Unlock P (Lock P M N sigma) N sigma) M)
```

where `isTrue: (term -> o) -> term -> Type` is the encoding in pure LF of the decision procedure checking if the predicate represented by `P` holds on the argument represented by `N`. The rest of the encoding presented in Section 2.1 remains unchanged.

3. Non-Commutative Linear Logic

In substructural logics, some rules are subject to *side conditions* and structural constraints on the shape of assumptions or premises. In this section, we outline an encoding in $\mathsf{LF}_{\mathcal{P}}$ of a particular substructural logic, namely Non-Commutative Linear Logic [26]. We briefly recall the syntax and the rules of NCLL, before engaging the task of encoding it in $\mathsf{LF}_{\mathcal{P}}$ (see Section 3.1).

First we introduce the ordered fragment of NCLL[2]. For conciseness, but not for political inclinations, we discuss only *right-ordered implication* $\twoheadrightarrow$ whose introduction/elimination rules are:

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

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

The gist of these rules is that "ordered assumptions occur exactly once and in the order they were made". The linear fragment of NCLL amounts to the following rules :

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

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

Finally, the intuitionistic fragment of NCLL is given by:

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

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

In order to illustrate how to deal with further complexities, we introduce also *ordered conjunction*:

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

$$\frac{\Gamma;\Delta_2;\Omega_2 \vdash M : A \bullet B \quad \Gamma;\Delta_1;(\Omega_1,z{:}A,z'{:}B,\Omega_3) \vdash N : C}{\Gamma;(\Delta_1 \bowtie \Delta_2);(\Omega_1,\Omega_2,\Omega_3) \vdash \mathit{let}\ z \bullet z' = M\ \mathit{in}\ N : C}$$

3.1 NCLL in $\mathsf{LF}_{\mathcal{P}}$

The main reason for the choice of this case study is that NCLL is particularly problematic to encode naïvely in a type theory-based LF, due to the fact that it effectively needs to access the derivation context, which, clearly, is not available at the object level (for a more detailed discussion see, *e.g.*, [13]).

Our encoding is a *shallow* one in the sense that we do not encode explicitly the proof terms of the original system but represent only types as formulæ. The idea of our encoding is *not* to make any *a priori* distinction between intuitionistic, linear, and order variables. It will be the order, linear, and intuitionistic *introduction rules* which will *canonize* them in those roles, if the constraints enforced by the appropriate $\mathcal{P}$'s in the locks will filter them out as suitable. We start by encoding right-ordered implication, see Section 3. The information necessary to check the condition on the occurrence of z, as the last variable in the ordered context, can be extracted by *local inspection* of the proof term. As it is the case in any LF-based logical framework the proof term fully records all previous derivation steps. Hence, we can introduce in $\mathsf{LF}_{\mathcal{P}}$ suitable *well-behaved* predicates in order to filter out proof terms satisfying such constraints. The encodings of rules $(\twoheadrightarrow I)$ and $(\twoheadrightarrow E)$ are:

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

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

where `impRight:o->o->o` represents the $\twoheadrightarrow$ constructor of *right-ordered implications*, and `Rightmost(`$\Gamma\vdash_\Sigma$ `M:(True A) -> (True B))` checks that:

1. `M` is an abstraction (*i.e.*, `M` $\equiv \lambda$`z:(True A).M'`);
2. all free variables in `M` occur in subterms whose type is either `o` or `(True A)` for some `A:o`;
3. the bound variable `z` occurs only once and never to the right of a variable bound by an abstraction which is the third argument of `impRightIntro` in the normal form of `M'`;
4. the bound variable `z` does not occur in the normal form of `M'`, in the fourth argument of the `impElim` and `impLinearElim` constructors.

The second check is necessary for the predicate to well-behaved, namely closed under substitution. The reference to the normal form could be dropped if we utilize $\mathsf{LF}_{\mathcal{P}}$ in canonical form [20]. The third check ensures linearity and the right ordering of the ordered variables, which in our setting are those bound by a $\lambda^{>}$. The last constraint is due to the presence of the intuitionistic and linear fragments of NCLL. In order to avoid the failure of the normalization step in the logic, the rule $(\twoheadrightarrow E)$ requires that the linear and ordered parts of the derivation context be empty in the second assumption

Linear implication is treated similarly by introducing a suitable predicate "ensuring" the correct introduction of the $\multimap$ constructor. Hence, the encodings of the rules $(\multimap I)$ and $(\multimap E)$ are as follows:

```
impLinearIntro: ΠA,B:o. ΠM:(True A) -> (True B).
                        L^{Linear}_{M,(True A)->(True B)}[True(impLinear A B)]
impLinearElim : ΠA,B:o.
          (True (impLinear A B)) -> (True A) -> (True B)
```

where `impLinear:o->o->o` represents the $\multimap$ constructor of *linear implications* and `Linear(`$\Gamma\vdash_\Sigma$ `M:(True A)->(True B))` holds if:

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

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

```
impIntro: ΠA,B:o. ΠM:(True A) -> (True B).(True (imp A B))
impElim : ΠA,B:o. (True(imp A B)) -> (True A) -> (True B)
```

Notice that in the encodings of rules $(\twoheadrightarrow E)$ and $(\multimap E)$ we have not enforced any conditions on the free variables occurring in terms, Indeed, the obvious requirements surface in the adequacy theorem which will be discussed in Section 4.

Ordered conjunction can be encoded with the same philosophy, but for the remark that in this case it is the *elimination rule* that acts at *abstraction-time* and hence needs a lock mechanism to filter out bad pattern matching constructions in the *let* operator. On the other hand the introduction rule can be encoded straightforwardly:

```
ordConjIntro: ΠA,B:o.
          (True A) -> (True B) -> (True (ord_conj A B))
```

```
ordConjElim:
    ΠA,B,C:o.
    ΠM :(True (ord_conj A B)).
    ΠN :(True A) -> (True B) -> (True C).
    Πh :(True A).
    Πh':(True B).L^{PM}_{[M,N,h,h']}[(True C)]
```

where the predicate $\mathcal{PM}$ (Pattern Matching) checks that:

1. all free variables in `M` and `N` occur in a subterm whose type is either `o` or `(True A)` for some `A:o`;
2. `M`$\equiv$`(ordConjIntro A B h h')`;
3. `N`$\equiv \lambda$`z:(True A).`λ`z':(True B).N'`;
4. `z` and `z'` occur once and in that order in the NF of `N'`, and do not occur within fourth arguments of the `impElim` and `impLinearElim` constructors.

As far as we know, this is the first example of an encoding of non-commutative linear logic in an LF-like framework. External predicates could be simplified if we utilize $\mathsf{LF}_{\mathcal{P}}$ in canonical form.

4. The adequacy of the adequacy

Is the classical notion of adequacy adequate? According to the seminal work [17], the notion of adequacy formally amounts to a *compositional bijection* between some tokens of the object system, usually including the language of theorems, and the notion of proof, and the *canonical forms* of some suitable types of the Logical Framework. The very notion of canonical form has triggered a whole new and insightful style of presentation of Logical Frameworks [18, 28].

However, there are cases where the formulation of the compositionality requirement is problematic, see *e.g.* [13, 20], because of the ambiguous use of the metalanguage free variables, *i.e.* as object level free variables, as well as schematic metavariables, and assumption witnesses. We think that a very simple amendment to the standard definition can overcome some controversies which arise in the exact formulation of the adequacy statement as far as *assumptions*. Our proposal is that not merely canonical forms should be the target of the encoding but canonical forms which express *morphisms*, in the style of categorical semantics, *i.e.* the *functional closures of the canonical forms* in some appropriate sense. Thus, object level *hypothetical judgements* would have an explicit formulation in the adequacy theorem as *metalanguage morphisms*, and the use of the term "compositionality", which pertains to morphisms, would be substantiated.

The adequacy for the encoding of NCLL in $\mathsf{LF}_{\mathcal{P}}$ in Section 3.1 provides a case-in-point. By the definition stipulated above, we can enforce on all assumptions the required constraints.

THEOREM 3 (Adequacy for NCLL in $\mathsf{LF}_{\mathcal{P}}$). *Let* $\mathcal{X} = \{P_1, \ldots, P_n\}$ *be a set of atomic formulæ occurring in formulæ* $A_1, \ldots, A_k, A$. *Then, there exists a bijection between derivations of the judgment* $(A_1, \ldots, A_{i-1}); (A_i, \ldots, A_{j-1}); (A_j, \ldots, A_k) \vdash A$ *in non-commutative linear logic, and proof terms* `h` *in* η*-long normal form such that we can derive the judgment in Figure 2, where the variables* $\mathtt{h_i}, \ldots, \mathtt{h_{j-1}}$ *occur in* `h` *only once,* $\mathtt{h_j}, \ldots, \mathtt{h_k}$ *occur in* `h` *only once and, precisely, in this order, and* $\Gamma_{\mathcal{X}}$ *is the context* `P1:o,...,Pn:o` *representing the object-language propositional formulæ* $P_1, \ldots, P_n$. ■

The above statement of adequacy can look complicated and the typing judgment in Figure 2 might look scaring. However, if we allow to introduce in the metalanguage suitable *linear* and *ordered* λ- and Π-binders, we obtain the following rephrasing which should clarify the meaning of *functional closures of the canonical forms*:

$\Gamma_{\mathcal{X}} \vdash \lambda \mathtt{h}_1{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_1)).\ldots.\lambda \mathtt{h}_{i-1}{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_{i-1})).$
$(\mathtt{impLinearIntro}\ \epsilon_{\mathcal{X}}(A_i)\ (\ldots)\ \lambda \mathtt{h}_i{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_i)).\ldots$
$\ldots(\mathtt{impLinearIntro}\ \epsilon_{\mathcal{X}}(A_{j-1})\ (\ldots)\ \lambda \mathtt{h}_{j-1}{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_{j-1})).$
$(\mathtt{impRightIntro}\ \epsilon_{\mathcal{X}}(A_j)\ (\ldots)\ \lambda \mathtt{h}_j{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_j)).\ldots.$
$\ldots(\mathtt{impRightIntro}\ \epsilon_{\mathcal{X}}(A_k)\ \epsilon_{\mathcal{X}}(A)\ \lambda \mathtt{h}_k{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_k)).\mathtt{h})\ldots))\ldots) :$
$\Pi \mathtt{h}_1{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_1)).\ldots.\Pi \mathtt{h}_{i-1}{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_{i-1})).$
$(\mathtt{True}\ (\mathtt{impLinear}\ \epsilon_{\mathcal{X}}(A_i)\ldots(\mathtt{impLinear}\ \epsilon_{\mathcal{X}}(A_{j-1})$
$(\mathtt{impRight}\ \epsilon_{\mathcal{X}}(A_j)\ldots(\mathtt{impRight}\ \epsilon_{\mathcal{X}}(A_k)\ \epsilon_{\mathcal{X}}(A))\ldots))\ldots))$

Figure 2. The adequacy judgment

$\Gamma_{\mathcal{X}} \vdash \lambda \mathtt{h}_1{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_1)).\ldots.\lambda \mathtt{h}_{i-1}{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_{i-1})).$
$\hat{\lambda} \mathtt{h}_i{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_i)).\ldots.\hat{\lambda} \mathtt{h}_{j-1}{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_{j-1})).$
$\lambda^{>} \mathtt{h}_j{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_j)).\ldots.\lambda^{>} \mathtt{h}_k{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_k)).\mathtt{h}) :$
$\Pi \mathtt{h}_1{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_1)).\ldots.\Pi \mathtt{h}_{i-1}{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_{i-1})).$
$\hat{\Pi} \mathtt{h}_i{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_i)).\ldots.\hat{\Pi} \mathtt{h}_{j-1}{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_{j-1})).$
$\Pi^{>} \mathtt{h}_j{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_j)).\ldots.\Pi^{>} \mathtt{h}_k{:}(\mathtt{True}\ \epsilon_{\mathcal{X}}(A_k)).(\mathtt{True}\ \epsilon_{\mathcal{X}}(A))$

The moral of these encodings is that, in accordance to the purest LF philosophy, we need to check even unorthodox side-conditions only when we abstract variables from the context, *i.e.* when we *eliminate assumptions*, be it the case of introduction rules, as for non-commutative implication, or in elimination rules as for the ordered conjunction. The fact that this *abstraction-time protocol* is enough, supports the general tenet underpinning LF - that side-conditions are ultimately *local properties* of proofs.

4.1 The depth of shallowness

In the quest for the most *suitable* adequate encoding, where suitable may stand for *effective*, *elegant*, *efficient etc.*, we are usually faced with a large number of options. These have been often classified, generically and arbitrarily, as *deep* and *shallow* encodings. The terminology goes back to [7], where a deep encoding was defined as "representing syntax as a type within a mechanized logic".

Today, we express the difference between various representations of the same object language according to the amount of machinery delegated to the metalanguage, *i.e.*, *how close* (how shallow), or *how far* (how deep) the encoding is w.r.t. the logical framework taken into consideration. A "shallow encoding" aims at delegating to the logical framework as much as possible the notions and mechanisms (*e.g.*, α-conversion, capture-avoiding substitution, logical consequence relations etc.) of the object language. This approach is valuable from the practical point of view because it yields more concise and transparent encodings which do not impose on the user extra burdens, at the risk of losing the advantages of the object systems. Furthermore, and somewhat paradoxically, a shallow encoding often offers a deeper insight on the object system itself, because it poses standardization questions. The use of HOAS is a case-in-point of a shallow encoding of a language with binders.

However, when proving metatheoretic properties of the object language, shallow encodings capitalizing on HOAS are pushed to their limits and one needs to introduce appropriate axioms for reifying the properties of the metalanguage itself, see *e.g.* the case of the π-calculus [22, 27]. See also [15], on how to prove properties involving free variables when using HOAS. Therefore, deeper encodings of α-conversion are used in most metatheoretic treatments [16].

We suggest that $\mathsf{LF}_{\mathcal{P}}$ can be useful in addressing the issue of deep and shallow encodings of a given object system. Considering shallow the encodings carried out in $\mathsf{LF}_{\mathcal{P}}$ and deep the corresponding representations in traditional LF we can try to express results of the form:

PARADIGM 1. *Given an* $\mathsf{LF}_{\mathcal{P}}$ *signature* Σ*, where* $\mathcal{P}_1, \ldots, \mathcal{P}_n$ *is the list of the external predicates occurring in it, if the latter are* LF*-encoded by* $LF(\mathcal{P}_i)$ *respectively, calling* Σ' *the* LF*-signature and* Γ' *the typing context obtained by*

- *adding to* Σ $LF(\mathcal{P}_1), \ldots, LF(\mathcal{P}_n)$*;*
- *substituting occurrences of* $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\tau]$ *by* $(LF(\mathcal{P})\ N\ \sigma) \to \tau$*;*

then we have that for each $\mathsf{LF}_{\mathcal{P}}$*-derivation* $\Gamma \vdash_{\Sigma} M : \sigma$ *where there are no locks neither in* M *nor in* σ*, there is a corresponding* LF*-derivation* $\Gamma' \vdash_{\Sigma'} M' : \sigma'$*.*

This correspondence relies on the possibility of representing[3] in LF the external predicates used in $\mathsf{LF}_{\mathcal{P}}$. Notice that also M and σ, above, may undergo some changes in the "translation" carried out in LF. This is why we speak of the existence of a "corresponding" LF-derivation and use M' and σ' as the new subject and object, respectively, of the LF-typing judgment. The reason is that to "implement" in LF the lock-predicates in $\mathsf{LF}_{\mathcal{P}}$, which check constraints on variables and terms, we might need a deeper representation of the logic. Often we need to view the logic as a *generalized type-assignment system* and hence use as basic judgement "typing" rather than "truth", and introduce in the signature as explicit syntactic categories *e.g.* object logic proof-terms and variables. In $\mathsf{LF}_{\mathcal{P}}$ these are delegated to the metalanguage, or "swept under the rug" and dispatched to external oracles.

As an example of paradigm correspondence, consider the following two representations of the intuitionistic and linear fragments of NCLL. The "shallow" one is the encoding in Section 3.1. The "deep" one is obtained by viewing NCLL as a type assignment system and by implementing the external predicates over the *reification* at object level of the proof terms which were left to the metalanguage in $\mathsf{LF}_{\mathcal{P}}$. This encoding is essentially the one of [13] for linear logic. Notice, for instance, that the judgment `linear:(term->term)->Type` is morally the reification of our external predicate `Linear` used in the previous section. We conjecture that the same can be done also for the ordered fragment.

But this is not the end of the story. Even the LF encodings above may be considered shallow w.r.t. one based on *e.g.*, de Bruijn indices, first-order syntax, etc. Much more work needs to be done in order to make correspondences between different encodings more precise, which really means more *compositional*.

5. Comparing $\mathsf{LF}_{\mathcal{P}}$ with other systems

$\mathsf{LF}_{\mathcal{P}}$ is sufficiently inconspicuous to allow for neat highlighting of the peculiarities of other Logical Frameworks, underpinning alternate formal proof cultures, which have emerged in recent years.

5.1 $\mathsf{LF}_{\mathcal{P}}$ vis-à-vis $\lambda\Pi$-calculus modulo

The expressive power of $\lambda\Pi$-calculus modulo [12] is unlimited, given strong enough rewrite rules, in that it can radically change types in a derivation. For this reason, decidability and subject reduction cannot be proven in general for $\lambda\Pi$-calculus modulo. For instance, putting $\sigma \equiv \sigma \to \sigma$ one can type all terms of λ-calculus. In this sense, it is very close to an *intersection types* discipline. Actually, if the modulo relation is not taken to be symmetric, it can offer a very intriguing version *à la* Church of $\lambda\cap$ [4] , which is usually presented only *à la* Curry. In [12], an encoding of the classical PTSs of the λ-cube, and in particular of the Calculus of Constructions, is given, which can be extended to all GTS's [2].

$\mathsf{LF}_{\mathcal{P}}$, on the other hand, can only *freeze* types, possibly releasing them syntactically *unchanged*, under suitable circumstances.

In order to encode $\mathsf{LF}_{\mathcal{P}}$ in $\lambda\Pi$-calculus modulo we can *reify* the semantics of the lock-operator at the level of types by introducing in the signature a type $Pred$ and suitable constants of that type to

[3] This brings about yet another kind of adequacy that is *formally* irreducible in the sense of the Münchausen's Trilemma.

represent the external predicate, and a type constructor

$$lockT : \Pi P{:}Pred.\Pi\sigma{:}Type.\Pi N{:}\sigma.\Pi\tau{:}Type.Type$$

subject to the rewriting rule:

$$(lockT\ P\ \sigma\ N\ \tau) \longrightarrow^{\Gamma,Type} \tau$$

To represent the locking and unlocking of terms we need a deeper encoding since the conversion rules of $\lambda\Pi$-calculus modulo only appear at the level of types and kinds. In order to represent the $\lambda\Pi$-calculus modulo in $\mathsf{LF}_{\mathcal{P}}$, we need a deep, albeit straightforward encoding:

```
raw_term  : Type
type,kind : raw_term
pi,lam    : raw_term -> (raw_term -> raw_term) -> raw_term
app       : raw_term ->  raw_term -> raw_term
typing    : raw_term ->  raw_term -> Type
```

All typing rules of $\lambda\Pi$-calculus modulo can then be represented as suitable axioms involving the `typing` judgment. The conversion rule (*Conversion* $A \equiv_{\beta\mathcal{R}} B$) of [12] is rendered as:

```
conv: ΠA,B,t:raw_term.
 (typing A type) -> (typing B type) -> (typing t A) ->
 L^{≡βR}_⟨A,B⟩[(typing t B)]
```

where the external predicate $\equiv_{\beta\mathcal{R}}$ checks if `A` and `B` represent the encodings of two types A and B such that $A \equiv_{\beta\mathcal{R}} B$ holds, according to the rewriting rules. The modularity of $\mathsf{LF}_{\mathcal{P}}$ reflects exactly the modularity of $\lambda\Pi$-calculus modulo; our external predicate mechanism allows to plug-in the formal counterpart of the rewriting rules which, in turn, have been specified and plugged-in in the general framework of $\lambda\Pi$-calculus modulo.

5.2 $\mathsf{LF}_{\mathcal{P}}$ vis-à-vis Conditional LF

$\mathsf{LF}_{\mathcal{P}}$ is a direct descendant of the Conditional LF [21]. In the latter, the mechanism of *freezing* a type was achieved by not allowing immediate substitution in the application rule. The idea of utilizing *external* predicates to activate substitutions is already there, but the freezing mechanism does not have corresponding term constructors. Thus, the proof terms of Conditional LF do not record all the history of the derivation, while $\mathsf{LF}_{\mathcal{P}}$ does indeed *record* the recourse to the external oracle in the proof terms. Apart from this, the two systems are essentially inter-encodable in a shallow way.

5.3 $\mathsf{LF}_{\mathcal{P}}$ vis-à-vis Pattern LF

The Pattern LF [19] is related both to GLF [19] as well as the rewriting calculi of the ρ-cube [6, 10]. It supports pattern-matching application. Thus, it allows both for some *freezing* of types, as $\mathsf{LF}_{\mathcal{P}}$, and to *shortcut* the proofs of correctness of computations, in line with Poincaré's Principle [3] and rewrite-based Logical Frameworks [12]. As usual, a deep encoding in $\mathsf{LF}_{\mathcal{P}}$ of the Pattern LF can be given by viewing it as a typing system. A shallow encoding is problematic, given the lack of a pattern-matching mechanism in $\mathsf{LF}_{\mathcal{P}}$. It would be intriguing to pursue a comparison between the pattern matching mechanisms of typed rewriting calculi [12], such as the Pattern LF and the Concurrent LF [28].

5.4 $\mathsf{LF}_{\mathcal{P}}$ vis-à-vis Linear LF and Concurrent LF

More work needs to be carried out in order to assess encodings of LLF [9] and CLF [28] in $\mathsf{LF}_{\mathcal{P}}$. These will be satisfactory to the amount by which we will understand whether what needs to be checked, be it linearity, order, can be effectively achieved at *abstraction time*, namely - is it ultimately a *local property*?

References

[1] A. Avron, F. Honsell, I. A. Mason, and R. Pollack. Using typed lambda calculus to implement formal systems on a machine. *Journal of Automated Reasoning*, 9:309–354, 1992.

[2] H. Barendregt. Lambda Calculi with Types. In *Handbook of Logic in Computer Science*, vol. II, pp 118–310. Oxford UP, 1992.

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

[4] H. Barendregt, W. Dekkers, R. Statman. Perspectives in Logic: Lambda Calculus with Types. Cambridge UP, ISBN-13: 9780521766142, 2013

[5] B. Barras. Coq en Coq. RR-3026, INRIA, 1996, Projet COQ.

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

[7] R. Boulton, A. Gordon, M. Gordon, J. Harrison, J. Herbert, and J. Van Tassel. Experience with embedding hardware description languages in HOL. See Stavridou, Melham, and Boute (1992), pp 129–156.

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

[9] I. Cervesato, and F. Pfenning. A linear logical framework. Logic in Computer Science, 1996. In Proc. of LICS'96.

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

[11] T. Coquand, and G. Huet. The calculus of constructions. (1986).

[12] D. Cousineau and G. Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In Proc. of *TLCA*, vol. 4583 of *LNCS*, pp 102–117. Springer Berlin Heidelberg, 2007.

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

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

[15] J. Despeyroux, A. Felty, and A. Hirschowitz Higher-order abstract syntax in Coq, In *TLCA'95*, vol. 902 of *LNCS*, 1995

[16] A.D. Gordon, and T. Melham. Five Axioms of Alpha-Conversion. Theorem proving in higher order logics. Springer Berlin Heidelberg, 1996. 173-190.

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

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

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

[20] F. Honsell, M. Lenisa, L. Liquori, P. Maksimovic, and I. Scagnetto. LF_P – a logical framework with external predicates. In *Proc. of LFMTP 2012*. ACM Digital Library.

[21] F. Honsell, M. Lenisa, L. Liquori, and I. Scagnetto. A conditional logical framework. In *LPAR'08*, vol. 5330 of *LNCS*, pp 143–157, 2008.

[22] A. Bucalo, M. Hofmann, F. Honsell, M. Miculan, and I. Scagnetto. Consistency of the theory of contexts. Journal of Functional Programming, Vol. 16, Issue 03, May 2006, pp 327-395.

[23] F. Pfenning, and C. Schuermann. Twelf user's guide, version 1.2. Tech. Rep. CMU-CS-98-173, Carnegie Mellon University, 1998.

[24] B. Pientka and J. Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In *Automated Reasoning*, vol. 6173 of *LNCS*, pp 15–21, 2010.

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

[26] J. Polakow, F. Pfenning. Natural deduction for intuitionistic non-commutative linear logic. *TLCA'99*, 1581, *LNCS*, pp 644–644, 1999.

[27] C. Röckl, D. Hirschkoff, and S. Berghofer. HOAS with induction in Isabelle/HOL: Formalizing the π-calculus and mechanizing the theory of contexts. In *FoSSaCS*, p.364-378, vol. 2030 of *LNCS* 2001.

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