25 Years of Formal Proof Cultures

F. Honsell

\{M,\}uni\{cipal,vers\}ità di Udine, Italy
sindaco@comune.udine.it

A scientific meeting in honor of Pierre-Louis Curien
Venezia September 9, 2013

joint work with M. Lenisa\(^1\), L. Liquori\(^2\), P. Maksimovic\(^2,3\) and I. Scagnetto\(^1\)
Università di Udine, Italy\(^1\)
Institut National de Recherche en Informatique et en Automatique, France\(^2\)
Mathematical Institute of the Serbian Academy of Sciences and Arts, Serbia\(^3\)
Outline

1. A brief history of Proof Cultures
2. The Framework
3. Pragmatics
4. Conclusions
HAPPY BIRTHDAY PIERRE-LOUIS

Pierre-Louis Curien

Categorical Combinators

Information and Control archive

Volume 69 Issue 1-3, April/May/June 1986
Science and democracy: two of a kind

An analogy between two methods

- "promoting reason in the world is much more an ethical issue than it is a metaphysical issue, besides being our duty, and our only hope” Christine Korsgaard
- enlightenment, the role of reason, of critical dialogue
- democracy, government through public debate (transparency, accountability, provability)
- democratic elections allow to replace governments, without destroying institutions, as would be the case if it were a tyrant to chase out of government, just like replacing an existing rational explanation by a better theory
- universal rights, universal laws, equity
- Sen: address questions of enhancing justice and removing injustice, rather than to offer resolutions of questions about the nature of perfect justice
Science and democracy: two of a kind

- The Moghul emperor Akbar used to say that: reason has priority over tradition because even to challenge reason you need to have reasons.
- Incremental enhancement - foster progress, advancement rather than struggle to achieve the perfect theory, or the perfect justice, or the perfect democracy, or the ultimate truth.
- Not necessary to eliminate all conflicting reasonable positions until only one is left.
- Promote justice and equity and eliminate injustice and inequity.
- Innovation, literacy, transnationality, universality.
Humour and mathematical proof: two of a kind

Both prize

- value conciseness
- capitilize paradigm shifts
- rely on *reductio ad absurdum*
- enjoy finding hidden assumptions
- play with ambiguities

A mathematician, half way through a proof, during a seminar, said "And this trivially holds". Only to add, after a few seconds of silence, somewhat to himself: "... but is this really trivial, here? ... Hmm ...".

He kept silent for 5 minutes.
And finally triumphantly exclaimed "Yes, it is indeed trivial!"
Euclid’s $\sum_{\text{Tol\v{c}\i\v{e}l\v{a}}}$

Logical arguments arose in Ancient Greece to counteract court evidence based on persuasion.
Proofs are a kind of repeatable evidence that can undergo, scientific, critical appraisal.
Explicit assumptions can be shared.
Proofs are built incrementally: validity is compositional.

- Euclid I.1: To construct an equilateral triangle on a given finite straight line. Why do the two circles intersect? Why does the construction lie on a plane? Why don’t the sides meet earlier in a common segment?
- Euclid I.4: If two triangles have two sides equal to two sides respectively, and have the angles contained by the equal straight lines equal, then they also have the base equal to the base, the triangle equals the triangle, and the remaining angles equal the remaining angles respectively, namely those opposite the equal sides. Superposition of segments is assumed to be a valid argument, Hilbert took it as a postulate.
Euclid VII. 1: When two unequal numbers are set out, and the less is continually subtracted in turn from the greater, if the number which is left never measures the one before it until a unit is left, then the original numbers are relatively prime. **Proved only for 3 iterations.**

Euclid IX. 20: Prime numbers are more than any multitude of assigned prime numbers. **Proved only for a multitude consisting of 3 primes.**

There is no **Induction.**

But there are proofs of **non-existence,** which require **arguments** and not just **computations** – G.Dowek.

A multitude appears to amount to 3. Probably this is due to the fact that in Greek, nouns and verbs have **dual,** as well as **plural** endings. **Diagrams** have a crucial role in Greek Mathematics. The axiomatic method enforces **formalization,** and opens up to **abstract models.**

Numbers are **measurable segments.**

There is a **logical dependency** between propositions.
Some fundamental questions


The question "What is a RULE?" deserves some attention

- epistemological issues: Wittgenstein, Kripke
- the problem of induction
- understanding a joke
- the story of Lewis Carroll "What the tortoise (taught-us) said to Achilles (Ah-kill-ease)" : "Just do it". An infinite regress.
- do we need a proof that the task has been fulfilled, besides the very fact?
- Resort to sociology to explain how rules are understood. Just synthetic not even a priori.
- Yet another process irreducible to direct formalization.
- Proofs are a reductio ad unum of all rules, all evidence: understand one understand all.
Antonio Gramsci SPN, 323-43 (Q1112), 1932

In acquiring one's conception of the world one always belongs to a particular grouping which is that of all the social elements which share the same mode of thinking and acting. We are all conformists of some conformism or other, always *man-in-the-mass* or *collective* man. The question is this: of what historical type is the conformism, the mass humanity to which one belongs?

Conformismo significa niente altro che 'socialità', ma piace impiegare la parola 'conformismo' appunto per urtare gli imbecilli.

L’individuo è originale storicamente quando dà il massimo di risalto e di vita alla socialità.

Amatya Sen: Some degree of *conformism, (conventionality)* is necessary to make understanding possible in any field. Triple approach, anthropological approach.
In the history and the practice of Mathematics, both Western and not, from Archimedes to Bhaskaracarya, deductive proofs are not frequent. There are many traditions of rational enquiry and alternate proof cultures.

Formal proofs have been often criticized for providing evidence but not explanations. In practice hypothetico/deductive method was and is often replaced by heuristics, and rules-of-thumb, by conjectures subject to counterexamples, and even empirical methods.
A plurality of Proof Cultures?

Since Euclid first crystallized the concept of rigorous proof using the axiomatic/deductive method, philosophers have been questioning the nature of mathematics: is it analytic or synthetic? Algebra vs. Geometry

E.g. prove that all the othogonals carried through the midpoints of each side of a triangle meet in a single point.

Modern readings of this opposition are

- deduction from axioms vs. computation according to rules,
  Poincaré $2 + 2 = 4$, Shopenhauer $\frac{(7+9)\times 8}{3} - 2 = 42$
- proof checking vs. verification,
- proving inhabitability of judgements vs. definitional equality of types,
- proofs vs. demonstrations, $2 + 2 = 4$, $1729 = 1^3 + 12^3 = 10^3 + 9^3$; Bhaskara, Aryabhatya.
Alternate proof cultures

- **Physical analogies**, Archimedes’ Gedanken experiments in the Organon: Conceive a geometrical figure as composed of thin slices of a physical object hanging on a balance scale and subject to gravity.

  *E.g.* The segments joining the midpoints of each side of a triangle to the opposite vertex meet in a single point.

- **Leibniz**: Find the point of the plane of an acute triangle that has the smallest (weighted) sum of the distances to the vertices of the triangle.

- **Given a point inside a convex polyhedron, show that there exists a face of the polyhedron such that the projection of the point onto the plane of that face lies inside the face.**

- **Find the shortest path back to the stable that the rider should take so that the horse first drinks at the river and then grazes at the meadow.**

- **The towns A and B are separated by a straight river. In what place should we construct a bridge MN, (orthogonal) to the shores, to minimize the length of the road AMNB.**
A provocative question: what are formal proofs useful for?

- Proofs as first class citizen
- Proofs as repeatable experiments (security - proof carrying code)
- Proofs as explanations

BUT

- Münchausen trilemma by Hans Albert.
- Formal proofs are brittle
- Do formal proofs convey an intuitive understanding of the theorem? Why was it conjectured? Which heuristics was used?
- Are formal proofs for the human user or for the machine? What difference does it make if we do not understand the proof but have to rely on a machine rather than another form of authority?
- What difference is there between: to know that something is true vs to know why it is true?
If we ask of any knowledge: "How do I know that it’s true?", we may provide proof; yet that same question can be asked of the proof, and any subsequent proof. The Münchhausen trilemma is that we have only three options when providing proof in this situation:

- The circular argument, in which theory and proof support each other (i.e. we repeat ourselves at some point)
- The regressive argument, in which each proof requires a further proof, ad infinitum (i.e. we just keep giving proofs, presumably forever)
- The axiomatic argument, which rests on accepted precepts (i.e. we reach some bedrock assumption or certainty)
What does a proof that a program meets its specifications buy you?

- Having a proof is a sort of invariant, similar to the dimensions check in Physics, the divisibility by 9 test in decimal arithmetic.
- Proof checking is decidable although not theoretically feasible. It is super exponential.
- Proofs increase our trust in the program or statement.
- Greeks made very few mistakes indeed.
More provocative questions

- Are all **interesting** mathematical statements **easy** to prove: *e.g.* Karatsuba’s multiplication?
- Are long proofs of universal statements practically useless? In that all instances that one can encounter in practice are true **independently** of the long universal proof?
- Is there an epistemological value of computing special instances or counterexamples?
The Modern Quest for Computer Assisted Rigorous Proof

- Complex mathematical arguments
  - Kepler’s problem
  - 4-colours theorem

R. Milner: Is Computer Science (and Mathematics) an experimental science?

- DIGITAL WOES: the need for dependable certified software
  - the Pentium Bug – mass produced application
  - the failures of the ESA programme –
  - fly-by-wire – safety critical software

- a plethora of idiosyncratic formal systems for reasoning on computer systems

40 years ago De Bruijn’s AUTOMATH project started. But the last 25 years have harvested great successes Coq, LF, Elf, Alf... Cfr. Dowek: ”Les Métamorphoses du calcul”

Constructive proofs support program extraction of certified algorithms. E.g. Primes are infinite: quotient and remainder always exist; the largest
more than 30 years ago ……

A Framework for Defining Logics
R.Harper, F.Honsell, G.Plotkin
Journal of the ACM
Volume 40 Issue 1, Jan. 1993
Pages 143-184
In 2007 at the ACM/IEEE LICS Symposium LF received the Test-of-Time Award.

The Coq Proof Assistant

The four-color theorem, the Feit Thompson Theorem
Some LF and their implementations

<table>
<thead>
<tr>
<th>Logical Framework</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculus of Constructions ('88)</td>
<td>Coq, LEGO</td>
</tr>
<tr>
<td>Martin-Löf’s type theory ('92)</td>
<td>ALF, Agda</td>
</tr>
<tr>
<td>Edinburgh LF ('93)</td>
<td>LEGO, Twelf</td>
</tr>
<tr>
<td>$CC^{(Co)\text{Ind}}$ ('93–'95)</td>
<td>Coq, Matita</td>
</tr>
</tbody>
</table>

They differ on the kind of type theory they are based on. $CC^{(Co)\text{Ind}}$ is the most general one.
To build an ad hoc implementation for a specific formal system does not appear to be a worth task since:

- there are often several different presentations of the system and a wide range of equivalent notions;
- starting from scratch requires a daunting effort to build other tools such as the following:
  - support for binding operators and substitution;
  - mechanisms associated with proof checking and proof construction;
  - automated search, unification and pattern matching.

Instead we can use Logical Frameworks: metalogical formalisms which capture most of the features of a wide range of logics, once and for all.
The significance of the Edinburgh Logical Framework, LF

LF is a dependent simply typed $\lambda$-calculus. LF provides answers to all the previous fundamental questions concerning formal systems.

- Logics are modeled via the Judgements-as-Types paradigm.
- Wff are modeled by typed terms.
- Martin-Löf’s hypothetico-general judgements are represented by inhabited higher-order types.
- The evidence of a judgment, the fulfillment of a task or a specification, the satisfaction of an expectation, the proof of a theorem, are reduced to the inhabitation of a suitable type by a closed $\lambda$-term.
- A logic is encoded by a context-signature.

In LF there are only two ways of providing evidence:

- discovering that types are inhabited in a given signature;
- postulating that types are inhabited by introducing appropriate typed constants in the signature.
The Calculus of (Co)Inductive Constructions (CC\textsuperscript{(Co)Ind}) [Giménez, '95] is a Logical Framework which extends the “pure” *Calculus of Constructions* (CC) [Coquand-Huet, '88] with:

- primitive support for *inductive* and *coinductive* types (i.e., potentially infinite objects);
- principles which simplify formal proofs involving (co)inductive types (elimination, inversion, case analysis and the *Guarded Induction Principle*).

CC\textsuperscript{(Co)Ind} is implemented in the Coq system, developed at the INRIA since 1984, by the Coq Development Team (Christine Paulin-Mohring, Hugo Herbelin, . . . ).
The Coq system

Coq [?] is a proof assistant for higher-order logic. It is based on CC^{(Co)Ind} and provides:

- a logic metalanguage for formal specifications;
- a proof editing mode for interactive proof development (backward reasoning);
- a program extractor to build a real OCaml program from its formal specification.

Main features:

- functional languages constructs (case expressions, ML-like pattern matching, fixed point operators);
- a new tactic for circular proofs based on the Guarded Induction Principle.

The Coq system has received the ACM SIGPLAN Programming Languages Software 2013 Award.
Was everything settled? Formally proving software correct wrt its specifications.

Absolute certainty is a myth:
- the specification might be imprecise;
- the proof engine might be flawed - De Bruijn Principle.

The adequacy problem of every formalization. 
The formalization in itself, is irreducible to formalization. 
A potential infinite regress.

BUT, notwithstanding this

Formal Program correctness appears not to be used that much, 
AND YET
There have not been that many software failures. 
Why proving formally program correctness? Is it necessary?
The Open Logical Framework, $\text{LF}_P$

Since there exist alternate means for establishing evidence we tried to enlarge our notion of proof and introduced a logical framework with external predicates to accommodate such alternate means.

In $\text{LF}_P$ we introduce the machinery of locking/unlocking types, which allows for the formal opening of the Logical Framework to alternate means of providing evidence for judgements. We can incorporate evidence from computation engines or automated theorem proving, and symbolic computation or graphical tools based, or even intuitive arguments. The idea of $\text{LF}_P$ is to allow for conditions which need only to be recorded. Their verification can be delegated to an external proof engine, in the style of the Poincaré Principle of Barendregt et al and the Deduction Modulo of Dowek et al.

$\text{LF}_P$, vindicates all pragmatics “proof cultures” used in mathematics
Proving a computation correct vs. demonstrating its verification by directly executing it

Consider e.g. $1^1 \times 2^2 \times 3^3 = 108$.

Just simple arithmetics vs. reifying the rules for computing exponentials and products.

What guarantees that this formalization is adequate, i.e. it corresponds to our intended understanding of arithmetic?

Similarly, specifications, proof obligations do indeed correspond to their intended meanings and pragmatic usages?

No complete argument can ever be done completely internally to any system. We have to resort to some informal argument outside any possible De Bruijn Principle.

If one looks for a definitive proof of adequacy, one is led into an infinite regress.

MORAL: even the simplest application takes us outside the system. We have to “just do it” as in Carroll’s story or Munchhausen trilemma.
A brief history of “beta-blockers”

2001: The main idea of “freezing” reductions through predicates was first introduced in Cirstea-Kirchner-Liquori paper “The $\rho$-cube”, a collection, in Barendregts $\lambda$-cube stype of 8 type systems for the “Rho-calculus”, a Combinatory Reduction System (CLF) allowing also higher-order rewriting rule and rewriting rule applications. The syntax where $M, N ::= c | M \rightarrow N | MN$ and the reduction where $(M \rightarrow N)P \rightarrow_{\rho} \theta(N)$ with $\theta = Sol(M <\!\!< N)$. This was presented in Fossacs01.

2007: Plotkin Festschrift: The paper “A Framework for Defining Logical Frameworks” presents a first attempt to generalize pure LF into a family of frameworks, called GLF by extending the LF syntax with unary logical predicates $P(-), Q(-)$ and “predicate abstraction” $\lambda P:\Delta M$: the generalized untyped $\beta$-redex $(\lambda P:\Delta M) N$ reduces to $M\theta$, for a suitable substitution $\theta$ over $\text{Dom}(\Delta)$, provided the predicate $P$ holds on $N$.

- For example: the Pattern Logical Framework, PLF was an instance of GLF where patterns $P$ are first-order algebraic terms.

2008: LPAR: The paper “A Conditional Logical Framework” further extend LF into a new framework CLF by considering the predicates as “external oracles”: the new syntax emphasized this externalization by including now a “conditional abstraction” $\lambda P:\Delta M$ of conditional type $\Pi P:\Delta M$; a conditional typed $\beta$-redex $\Gamma \vdash (\lambda P:\Delta M)$ reduces to $M[N/x]$ provided the typed predicate $P(Fv(N) ; \Gamma \vdash N:\sigma)$ holds and redex and reduct have the same type.


The main idea: lock type constructors

- **Lock type constructors** are a sort of \(\Diamond\)-modality constructors for building types of the shape \(\mathcal{L}_{N,\sigma}[\rho]\), where \(\mathcal{P}\) is a predicate on the typed judgement \(\Gamma \vdash_{\Sigma} N : \sigma\).

- Following the standard specification paradigm in Constructive Type Theory, we define lock types using **introduction**, **elimination**, and **equality** rules:

  - **introduction rule:**
    \[
    \frac{\Gamma \vdash_{\Sigma} M : \rho \quad \Gamma \vdash_{\Sigma} N : \sigma}{\Gamma \vdash_{\Sigma} \mathcal{L}_{N,\sigma}[M] : \mathcal{L}_{N,\sigma}[\rho]}
    \]

  - **elimination rule:**
    \[
    \frac{\Gamma \vdash_{\Sigma} M : \mathcal{L}_{N,\sigma}[\rho] \quad \Gamma \vdash_{\Sigma} N : \sigma \quad \mathcal{P}(\Gamma \vdash_{\Sigma} N : \sigma)}{\Gamma \vdash_{\Sigma} \mathcal{U}_{N,\sigma}[M] : \rho}
    \]

  - **reduction rule (inducing an equality rule):**
    \[\mathcal{U}_{N,\sigma}[\mathcal{L}_{N,\sigma}[M]] \rightarrow_{\mathcal{L}} M\]
Example: rules of proof vs. rules of derivation

- Let us consider classical Modal Logics [e.g., K, KT, K4, KT4 (S4), KT45 (S5)] in Hilbert style.
- They all feature necessitation as a rule of proof.
- We can encode such a rule in LFₚ by locking the type True(□φ) as follows:

\[
\begin{align*}
o & : \text{Type} \\
□ & : o \rightarrow o \\
\text{True} & : o \rightarrow \text{Type} \\
\text{NEC} & : \Pi \phi : o. \Pi m : \text{True}(\phi). \mathcal{L}_{\text{Closed}}^{\text{Closed}}(m, \text{True}(\phi))[\text{True}(\square \phi)]
\end{align*}
\]

where \(\text{Closed}(\Gamma \vdash_{\Sigma} m : \text{True}(\phi))\) holds iff “all free variables occurring in \(m\) occur in a subterm of type \(o\)” (i.e., there are no free assumptions)
The Framework $\mathcal{L}_P$ (syntax)

$$\Sigma \in S \quad \Sigma ::= \emptyset \mid \Sigma, a:K \mid \Sigma, c:\sigma$$

Signatures

$$\Gamma \in C \quad \Gamma ::= \emptyset \mid \Gamma, x:\sigma$$

Contexts

$$K \in K \quad K ::= \text{Type} \mid \Pi x:\sigma. K$$

Kinds

$$\sigma, \tau, \rho \in \mathcal{F} \quad \sigma ::= a \mid \Pi x:\sigma. \tau \mid \sigma N \mid \mathcal{L}^P_{N,\sigma}[\rho]$$

Families

$$M, N \in \mathcal{O} \quad M ::= c \mid x \mid \lambda x:\sigma. M \mid M N \mid$$

$$\mathcal{L}^P_{N,\sigma}[M] \mid \mathcal{U}^P_{N,\sigma}[M] (| \text{let } x \leftarrow M \text{ in } N )$$

Objects

$\mathcal{L}_P$ is a purely first-order predicative type theory, adding locks and
unlocks to the vertex (1,0,0) of Barendregt’s cube.
The type system for \( \text{LF}_P \) proves judgements of the shape:

\[
\begin{align*}
\Sigma \quad \text{sig} & \quad \Sigma \text{ is a valid signature} \\
\Gamma \vdash \Sigma & \quad \Gamma \text{ is a valid context in } \Sigma \\
\Gamma \vdash \Sigma \quad K & \quad K \text{ is a kind in } \Gamma \text{ and } \Sigma \\
\Gamma \vdash \Sigma \quad \sigma : K & \quad \sigma \text{ has kind } K \text{ in } \Gamma \text{ and } \Sigma \\
\Gamma \vdash \Sigma \quad M : \sigma & \quad M \text{ has type } \sigma \text{ in } \Gamma \text{ and } \Sigma
\end{align*}
\]
Typing System (signatures, contexts, kinds)

**Signature rules**

\[
\begin{align*}
\frac{}{\emptyset \text{ sig }} (S\cdot \text{Empty}) & \quad \frac{\Sigma \text{ sig } \vdash \Sigma K \quad a \notin \text{Dom}(\Sigma)}{\Sigma, a:K \text{ sig }} (S\cdot \text{Kind}) \\
\frac{\Sigma \text{ sig } \vdash \Sigma \sigma:Type \quad c \notin \text{Dom}(\Sigma)}{\Sigma, c:\sigma \text{ sig }} (S\cdot \text{Type})
\end{align*}
\]

**Context rules**

\[
\begin{align*}
\frac{}{\vdash \Sigma \emptyset} (C\cdot \text{Empty}) & \quad \frac{\vdash \Sigma \Gamma \quad \Gamma \vdash \Sigma \sigma:Type \quad x \notin \text{Dom}(\Gamma)}{\vdash \Sigma \Gamma, x:\sigma} (C\cdot \text{Type})
\end{align*}
\]

**Kind rules**

\[
\begin{align*}
\frac{\vdash \Sigma \Gamma}{\Gamma \vdash \Sigma \text{Type}} (K\cdot \text{Type}) & \quad \frac{\Gamma, x:\sigma \vdash \Sigma K}{\Gamma \vdash \Sigma \Pi x:\sigma.K} (K\cdot \Pi)
\end{align*}
\]
Typing System (families)

Family rules

\[ \vdash_{\Sigma} \Gamma \vdash_{\Sigma} a : K \quad (F\cdot Const) \]

\[ \Gamma, x:\sigma \vdash_{\Sigma} \tau : \text{Type} \quad (F\cdot Pi) \]

\[ \Gamma \vdash_{\Sigma} \Pi x:\sigma.\tau : \text{Type} \]

\[ \Gamma \vdash_{\Sigma} \sigma : \Pi x:\tau.\text{K} \quad \Gamma \vdash_{\Sigma} N : \tau \]

\[ \Gamma \vdash_{\Sigma} \sigma . N : \text{K}[N/x] \quad (F\cdot App) \]

\[ \Gamma \vdash_{\Sigma} \rho : \text{Type} \quad \Gamma \vdash_{\Sigma} N : \sigma \quad (F\cdot Lock) \]

\[ \Gamma \vdash_{\Sigma} L_{N,\sigma}[\rho] : \text{Type} \]

\[ \Gamma \vdash_{\Sigma} \sigma : \text{K} \quad \Gamma \vdash_{\Sigma} K' \quad K \equiv_{\beta\ell} K' \]

\[ \Gamma \vdash_{\Sigma} \sigma : K' \quad (F\cdot Conv) \]
Typing System (objects)

Object rules

\[
\frac{\Gamma, \, \varepsilon}{\Gamma} \quad (O\cdot Const) \quad \frac{\Gamma \vdash c : \sigma \\ \Sigma}{\Gamma, \Sigma \vdash c : \sigma} \quad (O\cdot Var)
\]

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

\[
\frac{\Gamma \vdash M : \rho \quad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathcal{L}^P_{N, \sigma}[M] : \mathcal{L}^P_{N, \sigma}[\rho]} \quad (O\cdot Lock)
\]

\[
\frac{\Gamma \vdash M : \mathcal{L}^P_{N, \sigma}[\rho] \quad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathcal{U}^P_{N, \sigma}[M] : \rho} \quad (O\cdot Unlock)
\]

\[
\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash \tau : \text{Type}}{\Gamma \vdash M : \tau} \quad (O\cdot Conv)
\]
Main one-step $\beta\mathcal{L}$-reduction rules on “raw terms”

In $\text{LF}_P$, we have two types of reduction:

1. standard $\beta$-reduction

\[
(\lambda x: \sigma. M) N \rightarrow_{\beta\mathcal{L}} M[N/x] \quad (\beta \cdot \mathcal{O} \cdot \text{Main})
\]

2. $\mathcal{L}$-reduction.

\[
\mathcal{U}^P_{N,\sigma}[\mathcal{L}^P_{N,\sigma}[M]] \rightarrow_{\beta\mathcal{L}} M \quad (\mathcal{L} \cdot \mathcal{O} \cdot \text{Main})
\]

The latter allows for the dissolution of a lock, in the presence of an unlock.
The rules for closure under context of the one-step reduction are introduced as usual.

We denote the reflexive and transitive closure of $\rightarrow^\beta L$ by $\rightarrow^*\beta L$.

Also $\beta L$-definitional equality is defined in the standard way, as the reflexive, symmetric, and transitive closure of $\beta L$-reduction on kinds, families, and objects:

$\frac{T \rightarrow^\beta L T'}{T =^\beta L T'}$ \hspace{1cm} $(\beta L \cdot \text{Eq} \cdot \text{Main})$ \hspace{1cm}$\frac{T =^\beta L T}{T =^\beta L T'}$ \hspace{1cm} $(\beta L \cdot \text{Eq} \cdot \text{Refl})$

$\frac{T =^\beta L T'}{T' =^\beta L T}$ \hspace{1cm} $(\beta L \cdot \text{Eq} \cdot \text{Sym})$ \hspace{1cm}$\frac{T =^\beta L T'}{T' =^\beta L T''}$ \hspace{1cm} $(\beta L \cdot \text{Eq} \cdot \text{Trans})$
Well-behaved Predicates

Definition (Well-behaved predicates)

A finite set of predicates \( \{P_i\}_{i \in I} \) is well-behaved if each \( P \) in this set satisfies the following conditions:

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

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

Closure under reduction. If \( P(\Gamma \vdash_\Sigma N : \sigma) \) holds and \( N \rightarrow_{\beta_L} N' \) (\( \sigma \rightarrow_{\beta_L} \sigma' \)) holds, then \( P(\Gamma \vdash_\Sigma N' : \sigma) \) (\( P(\Gamma \vdash_\Sigma N : \sigma') \)) also holds.
Properties of LF

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

Confluence: $\beta\mathcal{L}$-reduction is confluent, i.e. if $T \rightarrow_{\beta\mathcal{L}} T'$ and $T \rightarrow_{\beta\mathcal{L}} T''$, then there exists a $T'''$, such that $T' \rightarrow_{\beta\mathcal{L}} T'''$ and $T'' \rightarrow_{\beta\mathcal{L}} T'''$.

Subject Reduction: If predicates are well-behaved, then:
1. If $\Gamma \vdash_{\Sigma} K$, and $K \rightarrow_{\beta\mathcal{L}} K'$, then $\Gamma \vdash_{\Sigma} K'$.
2. If $\Gamma \vdash_{\Sigma} \sigma : K$, and $\sigma \rightarrow_{\beta\mathcal{L}} \sigma'$, then $\Gamma \vdash_{\Sigma} \sigma' : K$.
3. If $\Gamma \vdash_{\Sigma} M : \sigma$, and $M \rightarrow_{\beta\mathcal{L}} M'$, then $\Gamma \vdash_{\Sigma} M' : \sigma$. 
Typing `let` objects

\[
\begin{align*}
\Gamma, x: \tau & \vdash \Sigma M : \mathcal{L}_{N, \sigma}^{P}[\rho] & \Gamma & \vdash \Sigma Q : \mathcal{L}_{N, \sigma}^{P}[\tau] \\
\Gamma & \vdash \Sigma \text{let } x \leftarrow Q \text{ in } M : \mathcal{L}_{N, \sigma}^{P}[\rho] \quad \text{let}
\end{align*}
\]

\[
\begin{align*}
\Gamma & \vdash \Sigma Q : \mathcal{L}_{N, \sigma}^{P}[\tau_1] & \Gamma, x: \tau_1 & \vdash \Sigma M : \mathcal{L}_{N, \sigma}^{P}[\tau_2] & \Gamma, y: \tau_2 & \vdash \Sigma M : \mathcal{L}_{N, \sigma}^{P}[\rho] \\
(\text{let } x \leftarrow Q \text{ in (let } y \leftarrow P \text{ in } M)) & = (\text{let } y \leftarrow P \text{ in (let } x = Q \text{ in } M))
\end{align*}
\]

\[
\begin{align*}
\Gamma, x: \tau & \vdash \Sigma M : \mathcal{L}_{N, \sigma}^{P}[\rho] & \Gamma & \vdash \Sigma Q : \tau \\
\Gamma & \vdash (\text{let } x \leftarrow \mathcal{L}_{N, \sigma}^{P}[Q] \text{ in } M) = M[x/Q]
\end{align*}
\]

\[
\begin{align*}
\Gamma & \vdash \Sigma Q : \mathcal{L}_{N, \sigma}^{P}[\tau] \\
\Gamma & \vdash (\text{let } x \leftarrow Q \text{ in } \mathcal{L}_{x, \sigma}^{P}[x]) = Q : \mathcal{L}_{N, \sigma}^{P}[\tau]
\end{align*}
\]
A monadic reading of $L_{N,\sigma}^P[\tau]$

- In the seminal paper [?] (awarded with the LICS Test-of-Time in 2009) the author gives a calculus based on a categorical semantics for computations, providing a correct basis for proving equivalence of programs, independent from any specific computational model.
- The main categorical notion behind this idea is that of monads:
- Once we have fixed a predicate $P$, and $N, \sigma$, we can think about the lock constructor as a closure operator, indeed a monad: the red-painting monad
  - whence, the let destructor and the relative rules we have seen descend directly from monads’ theory.
In order to concretely use LF\(_\mathcal{P}\) as a Logical Framework, we have to establish a notion of adequacy for object languages encodings.

With this aim in mind, we introduce a canonical version of LF\(_\mathcal{P}\), namely, CLF\(_\mathcal{P}\) (in the style of [?, ?]):

- this amounts to an extension of the standard \(\eta\)-rule with the clause 
  \[\mathcal{L}_{N,\sigma}^{\mathcal{P}}[U_{N,\sigma}^{\mathcal{P}}[M]] \rightarrow_{\eta} M,\]
  corresponding to the lock type constructor.
- The syntax of CLF\(_\mathcal{P}\) defines the normal forms of LF\(_\mathcal{P}\).
- . . . the typing system captures all of the judgements in \(\eta\)-long normal form derivable in LF\(_\mathcal{P}\).
- CLF\(_\mathcal{P}\) will be the basis for proving the adequacy of the encodings of our case studies.

Contrary to standard LF,

- not all of the judgements derivable in LF\(_\mathcal{P}\) admit a corresponding \(\eta\)-long normal form.
- . . . this is not the case when the predicates appearing in the LF\(_\mathcal{P}\) judgement are not satisfied in the given context.
- Nevertheless, CLF\(_\mathcal{P}\) is still powerful enough to obtain all relevant adequacy results.
Syntax and Type System for CLF

- Formation rules and judgements related to signatures, contexts, and kinds are the same as in LF.
- CLF Syntax is the following:
  \[\begin{align*}
  \alpha & \in A \quad \alpha ::= a \mid \alpha N \\
  \sigma, \tau, \rho & \in F \quad \sigma ::= \alpha \mid \Pi x:\sigma.\tau \mid L^P_{N,\sigma}[\rho] \\
  A & \in A_o \quad A ::= c \mid x \mid AM \mid U^P_{N,\sigma}[A] \\
  M, N & \in O \quad M ::= A \mid \lambda x:\sigma.M \mid L^P_{N,\sigma}[M]
  \end{align*}\]
  
  Atomic Families
  Canonical Families
  Atomic Objects
  Canonical Objects

- The latter together with the type system for CLF prove judgements of the shape:
  \[\begin{align*}
  \Gamma & \vdash \Sigma \sigma \text{ Type} \\
  \Gamma & \vdash \Sigma \alpha \Rightarrow K \\
  \Gamma & \vdash \Sigma M \leftarrow \sigma \\
  \Gamma & \vdash \Sigma A \Rightarrow \sigma
  \end{align*}\]

  \(\sigma\) is a canonical family in \(\Gamma\) and \(\Sigma\)
  \(K\) is the kind of the atomic family \(\alpha\) in \(\Gamma\) and \(\Sigma\)
  \(M\) is a canonical term of type \(\sigma\) in \(\Gamma\) and \(\Sigma\)
  \(\sigma\) is the type of the atomic term \(A\) in \(\Gamma\) and \(\Sigma\)

- The predicates \(P\) in CLF are defined on judgements \(\Gamma \vdash \Sigma M \leftarrow \sigma\).
- The hereditary substitution judgement \(T[M/x]^m_{\rho} = T'\) computes the normal form resulting from the substitution of one normal form into another.
Correspondence between LFₚ and CLFₚ judgments

Theorem (Correspondence)

Assume that all predicates in LFₚ are well-behaved. For any predicate P in LFₚ, we define a corresponding predicate in CLFₚ with: P(Γ ⊢ Σ M ⇐ σ) holds if Γ ⊢ Σ M ⇐ σ is derivable in CLFₚ and P(Γ ⊢ Σ M : σ) holds in LFₚ.

Then, we have:
1. If Σ sig is in η-Inf and is LFₚ-derivable, then Σ sig is CLFₚ-derivable.
2. If Γ ⊢ Σ Γ is in η-Inf and is LFₚ-derivable, then Γ ⊢ Σ Γ is CLFₚ-derivable.
3. If Γ ⊢ Σ K is in η-Inf, and is LFₚ-derivable, then Γ ⊢ Σ K is CLFₚ-derivable.
4. If Γ ⊢ Σ α : K is in η-Inf and is LFₚ-derivable, then Γ ⊢ Σ α ⇒ K is CLFₚ-derivable.
5. If Γ ⊢ Σ σ:Type is in η-Inf and is LFₚ-derivable, then Γ ⊢ Σ σ Type is CLFₚ-derivable.
6. If Γ ⊢ Σ A : α is in η-Inf and is LFₚ-derivable, then Γ ⊢ Σ A ⇒ α is CLFₚ-derivable.
7. If Γ ⊢ Σ M : σ is in η-Inf and is LFₚ-derivable, then Γ ⊢ Σ M ⇐ σ is CLFₚ-derivable.
Derivation rules of formal systems are often subject to side conditions.

Sometimes the latter are either rather difficult or impossible to encode naively in a type theory-based LF, due to:

- limitations of the LF,
- the fact that they need to access
  - the derivation context,
  - the structure of the derivation itself,
  - other structures and mechanisms not available at the object level.

Such side conditions can be handled by suitable external predicates in LF$_P$. 
Predicate Archetypes

We have isolated some patterns of predicates frequently occurring in the examples. The main archetype is that given constants or variables only occur

- with some modality $D$,
- in subterms satisfying the decidable property $C$.

Modalities can include any of the phrases such as:

- at least once,
- only once,
- the rightmost,
- does not occur,
- etc.

$C$ can refer to the syntactic form of the subterm or to that of its type, the latter being the main reason for allowing predicates in $\text{LF}_P$ to access the context.
Modal Logic \( S_4 \) in Hilbert style

\[
\begin{align*}
A_1 : & \quad \phi \to (\psi \to \phi) \\
A_2 : & \quad (\phi \to (\psi \to \xi)) \to (\phi \to \psi) \to (\phi \to \xi) \\
A_3 : & \quad (\neg \phi \to \neg \psi) \to ((\neg \phi \to \psi) \to \phi) \\
K : & \quad \Box (\phi \to \psi) \to (\Box \phi \to \Box \psi) \\
T : & \quad \Box \phi \to \phi \\
4 : & \quad \Box \phi \to \Box \Box \phi \\
MP : & \quad \phi, \phi \to \psi \quad \psi \\
NEC : & \quad \phi, \Box \phi
\end{align*}
\]

Here NEC is a rule of proof, i.e., it applies only to premises which do not depend on any assumption.

The idea is to use suitable lock types in rules of proof and “standard” types in the rules of derivation.
Encoding in $\mathbb{LF}_P$ of Modal Logic $S_4$ \textit{à la} Hilbert

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

True : o $\rightarrow$ Type

A1 : $\Pi \phi, \psi : o. \text{True}(\phi \rightarrow (\psi \rightarrow \phi))$

A2 : $\Pi \phi, \psi, \xi : o. \text{True}(\phi \rightarrow (\psi \rightarrow \xi)) \rightarrow (\phi \rightarrow \psi) \rightarrow (\phi \rightarrow \xi)$

A3 : $\Pi \phi, \psi : o. \text{True}((\neg \phi \rightarrow \neg \psi) \rightarrow ((\neg \phi \rightarrow \psi) \rightarrow \phi))$

K : $\Pi \phi, \psi : o. \text{True}(\Box(\phi \rightarrow \psi) \rightarrow (\Box \phi \rightarrow \Box \psi))$

T : $\Pi \phi : o. \text{True}(\Box \phi \rightarrow \phi)$

4 : $\Pi \phi : o. \text{True}(\Box \phi \rightarrow \Box \Box \phi)$

MP : $\Pi \phi, \psi : o. \text{True}(\phi) \rightarrow \text{True}(\phi \rightarrow \psi) \rightarrow \text{True}(\psi)$

NEC : $\Pi \phi : o. \Pi m : \text{True}(\phi). \mathcal{L}_{\text{Closed}_{m, \text{True}(\phi)}}[^{\text{True}(\Box \phi)}]$
Encoding NEC: the predicate $\text{Closed}$

$$\text{NEC} : \prod \phi : o . \prod m : \text{True}(\phi) . \mathcal{L}^{\text{Closed}}_{m, \text{True}(\phi)}[\text{True}(\square \phi)]$$

$\text{Closed}(\Gamma \vdash \Sigma \ m : \text{True}(\phi))$, holds iff “all free variables occurring in the proof term $m$ occur in a subterm of type $o$”.

The above predicate is well-behaved.
In LF\(_P\), one can also accommodate classical Modal Logics \(S_4\) and \(S_5\) in Natural Deduction style, as defined by Prawitz.

The latter have rules with rather elaborate restrictions on the shape of subformulae where assumptions occur.

\[
\frac{\Box \Gamma \vdash \phi}{\Box \Gamma \vdash \Box \phi} \quad (\Box I \cdot S_4) \quad \text{or} \quad \frac{\Delta \vdash \Box \Gamma \quad \Box \Gamma \vdash \phi}{\Delta \vdash \Box \phi} \quad (\Box I' \cdot S_4)
\]

\[
\frac{\Box \Gamma_0, \neg \Box \Gamma_1 \vdash \phi}{\Box \Gamma_0, \neg \Box \Gamma_1 \vdash \Box \phi} \quad (\Box I \cdot S_5)
\]
Encoding in LF of Modal Logics $S_4$ and $S_5$ à la Prawitz

BoxI : $\Pi\phi:o.\Pi m:\text{True}(\phi).\mathcal{L}^{\text{Boxed}}_{m,\text{True}(\phi)}[\text{True}(\Box\phi)]$

Again, the crucial role is played by a predicate, namely $\text{Boxed}(\ )$:

$S_4$: $\text{Boxed}(\Gamma \vdash \Sigma m: \text{True}(\phi))$ holds iff the occurrences of free variables of $m$ occur in subterms whose type has the shape $\text{True}(\Box\psi)$ or $o$;

$S_5$: $\text{Boxed}(\Gamma \vdash \Sigma m: \text{True}(\phi))$ holds iff occurrences of the free variables of $m$ occur in subterms whose type has the shape $\text{True}(\Box\psi), \text{True}(\neg\Box\psi)$ or $o$.

These predicates are well behaved.
Non-commutative Linear Logic (NCLL)

- In substructural logics, some rules are subject to *side conditions* and structural constraints on the shape of assumptions or premises.
- First we introduce the ordered fragment of NCLL:

\[
\frac{}{\Gamma; \cdot; z : A \vdash z : A} \quad (\text{ovar}) \quad \frac{\Gamma; \Delta; (z : A, \Omega) \vdash M : B}{\Gamma; \Delta; \Omega \vdash \lambda > z : A. M : A \multimap B} \quad (\multimap I)
\]

\[
\frac{\Gamma; \Delta_1; \Omega_1 \vdash M : A \multimap B \quad \Gamma; \Delta_2; \Omega_2 \vdash N : A}{\Gamma; (\Delta_1 \ltimes \Delta_2); (\Omega_1, \Omega_2) \vdash M \gt N : B} \quad (\multimap E)
\]

\[
\frac{\Gamma; \Delta; (z : A, \Omega) \vdash M : B}{\Gamma; \Delta; \Omega \vdash \lambda < z : A. M : A \multimap B} \quad (\multimap I)
\]

\[
\frac{\Gamma; \Delta_2; \Omega_2 \vdash M : A \multimap B \quad \Gamma; \Delta_1; \Omega_1 \vdash N : A}{\Gamma; (\Delta_1 \ltimes \Delta_2); (\Omega_1, \Omega_2) \vdash M \lt N : B} \quad (\multimap E)
\]

- The gist of these rules is that “ordered assumptions occur exactly once and in the order they were made”.

NCLL: the linear and intuitionistic fragments

- **Linear rules:**
  
  \[
  \frac{\Gamma; y : A; \cdot \vdash y : A}{\Gamma; y : A; \cdot \vdash y : A} \quad \text{(Ivar)}
  \]
  
  \[
  \frac{\Gamma; (\Delta, y : A); \Omega \vdash M : B}{\Gamma; \Delta; \Omega \vdash \lambda y : A. M : A \to B} \quad (\to \text{I})
  \]
  
  \[
  \frac{\Gamma; \Delta; \Omega \vdash M : A \to B \quad \Gamma; \Delta; \Omega \vdash N : A}{\Gamma; (\Delta_1 \Join \Delta_2); \Omega \vdash M \langle N : B \quad (\to \text{E})
  \]

- **Intuitionistic rules:**
  
  \[
  \frac{\Gamma_1, x : A, \Gamma_2; \cdot \cdot \cdot \vdash x : A}{(\Gamma_1, x : A, \Gamma_2); \cdot \cdot \cdot \vdash x : A} \quad \text{(ivar)}
  \]
  
  \[
  \frac{\Gamma, x : A; \Delta; \Omega \vdash M : B}{\Gamma; \Delta; \Omega \vdash \lambda x : A. M : A \to B} \quad (\to \text{I})
  \]
  
  \[
  \frac{\Gamma; \Delta; \Omega \vdash M : A \to B \quad \Gamma; \cdot \cdot \cdot \vdash N : A}{\Gamma; \Delta; \Omega \vdash MN : B} \quad (\to \text{E})
  \]
The Problems of Encoding NCLL in LF_{\mathcal{P}}

- NCLL is problematic to encode in a type theory-based LF, due to the fact that it needs to access the derivation context, which is not available at the object level [?].
- Our encoding is a shallow one, i.e., we represent only types as formulæ:
  - we do not make any a priori distinction between intuitionistic, linear, and ordered variables.
  - It will be the ordered, linear, and intuitionistic introduction rules which will canonize them in those roles, (according to the appropriate \mathcal{P}'s in the locks).
  - As it is the case in any LF-based logical framework the proof term fully records all previous derivation steps:
    - local inspection of the proof term will check the condition on the occurrence of z, as, e.g., the last variable in the ordered context.
    - Hence, we can introduce in LF_{\mathcal{P}} suitable well-behaved predicates in order to filter out proof terms satisfying such constraints.
(Shallow) Encoding in LF\(\mathcal{P}\) of NCLL: right-ordered fragment

\[
\text{impRightIntro: } \Pi A,B:o. \Pi M: (\text{True } A) \rightarrow (\text{True } B).
\]
\[
L_{\text{Rightmost } M, (\text{True } A) \rightarrow (\text{True } B)}[\text{True } (\text{impRight } A \ B)]
\]
\[
\text{impRightElim : } \Pi A,B:o. (\text{True } (\text{impRight } A \ B)) \rightarrow (\text{True } A) \rightarrow (\text{True } B)
\]

where \(\text{impRight}:o\rightarrow o\rightarrow o\) represents the \(\rightarrow\) constructor of right-ordered implications, and \(\text{Rightmost}(\Pi_{-\Sigma} M: (\text{True } A) \rightarrow (\text{True } B))\) checks that:

1. \(M\) is an abstraction (\(i.e., M \equiv \lambda z: (\text{True } A).M'\));
2. all free variables in \(M\) occur in subterms whose type is either \(o\) or \((\text{True } A)\) 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 \(\text{impRightIntro}\) 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 \(\text{impElim}\) and \(\text{impLinearElim}\) constructors.
(Shallow) Encoding in LF of NCLL: left-ordered fragment

\[\text{impLeftIntro} : \Pi A, B : o. \ \Pi M : (\text{True } A) \rightarrow (\text{True } B) .\]
\[\text{Leftmost} \ M, (\text{True } A) \rightarrow (\text{True } B) [\text{True } (\text{impLeft } A \ B)]\]

\[\text{impLeftElim} : \Pi A, B : o. (\text{True } A) \rightarrow (\text{True } (\text{impLeft } A \ B)) \rightarrow (\text{True } B)\]

- where \text{impLeft} : o \rightarrow o \rightarrow o represents the \(\rightarrow\) constructor of left ordered implications. Finally, \text{Leftmost}(\Gamma \vdash \Sigma M : (\text{True } A) \rightarrow (\text{True } B)) checks the same constraints of the \text{Rightmost} predicate.

- Notice that we switch the arguments of left ordered application in \text{impLeftElim}, in order to simplify the ordering constraints of the \text{Rightmost} and \text{Leftmost} predicates.
(Shallow) Encoding in LF$\mathcal{P}$ of NCLL: linear fragment

impLinearIntro: $\forall A,B:o. \forall M: (\text{True } A) \rightarrow (\text{True } B)$.  

\[ \mathcal{L}_{\text{Linear}}^M,(\text{True } A)\rightarrow(\text{True } B)[\text{True } (\text{impLinear } A B)] \]

impLinearElim : $\forall A,B:o. (\text{True } (\text{impLinear } A B)) \rightarrow (\text{True } A) \rightarrow (\text{True } B)$

where $\text{impLinear}:o\rightarrow o \rightarrow o$ represents the $\rightarrow o$ constructor of linear implications and $\text{Linear}(\Gamma \vdash \Sigma M: (\text{True } A) \rightarrow (\text{True } B))$ holds if:

1. $M$ is an abstraction (i.e., $M \equiv \lambda z: (\text{True } A). M'$);
2. all free variables in $M$ occur in subterms whose type is either $o$ or $(\text{True } A)$ 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.
(Shallow) Encoding in $\text{LF}_\mathcal{P}$ of NCLL: intuitionistic fragment

impIntro: $\Pi A,B:o. \Pi M:(\text{True } A) \to (\text{True } B).(\text{True } (\text{imp } A \text{ B}))$

impElim : $\Pi A,B:o. (\text{True } (\text{imp } A \text{ B})) \to (\text{True } A) \to (\text{True } B)$

- Notice that in the encodings of rules ($\to E$) and ($\rightarrow\circ E$) 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 the next slide.
Theorem (Adequacy for Non-Commutative Linear Logic)

Let $\mathcal{X} = \{P_1, \ldots, P_n\}$ be a set of atomic formulæ occurring in formulæ $A_1, \ldots, A_k, A$. Then, there exists a bijection between derivations of the judgment $(A_1, \ldots, A_{i-1}); (A_i, \ldots, A_{j-1}); (A_j, \ldots, A_k) \vdash A$ in non-commutative linear logic, and proof terms $h$ such that

$\Gamma_{\mathcal{X}}, h_1 : (\text{True } \epsilon_{\mathcal{X}}(A_1)), \ldots, h_k : (\text{True } \epsilon_{\mathcal{X}}(A_k)) \vdash h : (\text{True } \epsilon_{\mathcal{X}}(A))$ in $\eta$-long normal form, where the variables $h_i, \ldots, h_{j-1}$ occur in $h$ only once (but never in the fourth argument of $\text{impElim}$), $h_j, \ldots, h_k$ occur in $h$ only once and, precisely, in this order (but never in the fourth argument of $\text{impElim}$ or $\text{impLinearElim}$), and $\Gamma_{\mathcal{X}}$ is the context $P_1:o, \ldots, P_n:o$ representing the object-language propositional formulæ $P_1, \ldots, P_n$. 
The adequacy of the adequacy

Is the classical notion of adequacy adequate?

- According to [?], it formally amounts to a *compositional bijection* between some tokens of the object system and the *canonical forms* of some suitable types of the LF.
- The very notion of canonical form has triggered a whole new and insightful style of presentation of LFs [?, ?].

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.
Adequacy for NCLL revisited (I)

Theorem (Adequacy for NCLL in LFᵦ)

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 $\eta$-long normal form such that we can derive the following judgment:

$$\Gamma, \mathcal{X} \vdash \lambda h_1 : (\text{True} \in \mathcal{X}(A_1)) \ldots \lambda h_{i-1} : (\text{True} \in \mathcal{X}(A_{i-1})).$$

$$(\text{impLinearIntro} \in \mathcal{X}(A_i) \ldots) \lambda h_1 : (\text{True} \in \mathcal{X}(A_i)) \ldots$$

$$(\text{impRightIntro} \in \mathcal{X}(A_j) \ldots) \lambda h_j : (\text{True} \in \mathcal{X}(A_j)) \ldots$$

$$(\text{impRightIntro} \in \mathcal{X}(A_k) \in \mathcal{X}(A) \lambda h_k : (\text{True} \in \mathcal{X}(A_k)) . h \ldots) \ldots):$$

$$\Pi h_1 : (\text{True} \in \mathcal{X}(A_1)) \ldots \Pi h_{i-1} : (\text{True} \in \mathcal{X}(A_{i-1})).$$

$$(\text{True} \circ \text{impLinear} \in \mathcal{X}(A_i) \ldots) (\text{impLinear} \in \mathcal{X}(A_{j-1})$$

$$(\text{impRight} \in \mathcal{X}(A_j) \ldots) (\text{impRight} \in \mathcal{X}(A_k) \in \mathcal{X}(A)) \ldots) \ldots))$$

where the variables $h_i, \ldots, h_{j-1}$ occur in $h$ only once, $h_j, \ldots, h_k$ occur in $h$ only once and, precisely, in this order, and $\Gamma, \mathcal{X}$ is the context $P_1:o, \ldots, P_n:o$ representing the object-language propositional formulæ $P_1, \ldots, P_n$. 

□
The previous statement can look quite complicated.

However, if we introduce in the metalanguage suitable linear and ordered λ-binders as follows:

- \( \hat{\lambda} h_i : (\text{True} \in X(A_i)) \cdot M = \lambda x : (\text{True} \in X(A_i)) \cdot U_{h_i M}^{\text{Linear}} [L_{h_i M}^{\text{Linear}} [M]] \)
- \( \lambda > h_i : (\text{True} \in X(A_i)) \cdot M = \lambda x : (\text{True} \in X(A_i)) \cdot U_{h_i M}^{\text{Rightmost}} [L_{h_i M}^{\text{Rightmost}} [M]] \)
- \( \lambda < h_i : (\text{True} \in X(A_i)) \cdot M = \lambda x : (\text{True} \in X(A_i)) \cdot U_{h_i M}^{\text{Leftmost}} [L_{h_i M}^{\text{Leftmost}} [M]] \)

and similarly for Π-binders, we obtain the following rephrasing which should clarify the meaning of functional closures of the canonical forms:

\[
\Gamma \vdash \lambda h_1 : (\text{True} \in X(A_1)) \ldots \lambda h_{i-1} : (\text{True} \in X(A_{i-1})) \\
\hat{\lambda} h_1 : (\text{True} \in X(A_1)) \ldots \hat{\lambda} h_{j-1} : (\text{True} \in X(A_{j-1})) \\
\lambda > h_j : (\text{True} \in X(A_j)) \ldots \lambda > h_k : (\text{True} \in X(A_k)).h : \\
\Pi h_1 : (\text{True} \in X(A_1)) \ldots \Pi h_{i-1} : (\text{True} \in X(A_{i-1})) \\
\hat{\Pi} h_1 : (\text{True} \in X(A_1)) \ldots \hat{\Pi} h_{j-1} : (\text{True} \in X(A_{j-1})) \\
\Pi > h_j : (\text{True} \in X(A_j)) \ldots \Pi > h_k : (\text{True} \in X(A_k)).(\text{True} \in X(A))
\]

In accordance to the purest LF philosophy, we need to check unorthodox side-conditions only when we abstract variables from the context, i.e. when we eliminate assumptions.

The fact that this abstraction-time protocol is enough, supports the general tenet underpinning LF - that side-conditions are ultimately local properties of proofs.
Conclusions

In the next slides we conclude with some insights and conjectures about the following topics:

- shallow vs. deep encodings;
- the expressive power of LF\(_P\);
- comparisons of LF\(_P\) with other popular systems;
- directions for future work and investigations.
The “depth” of shallowness (I)

- Going back to \[\text{?}\], in the quest for the perfect encoding, we are usually faced with the dichotomy between deep and shallow encodings.
- In \[\text{?}\], a deep encoding was defined as “representing syntax as a type within a mechanized logic”.
- Today, we speak about the amount of machinery delegated to the metalanguage, \textit{i.e.}, how close (how shallow), or how far (how deep) the encoding is \textit{w.r.t.} the logical framework taken into consideration.
- A “shallow encoding” aims at delegating to the LF as much as possible the notions and mechanisms of the object language.
  - **pros**: more concise and transparent encodings, deeper insight on the object system itself, because it poses standardization questions (\textit{e.g.}, use of HOAS for languages with binders).
  - **cons**: proving metatheoretic properties of the object language can be a nightmare.
The “depth” of shallowness (II)

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

**Paradigm**

Given an $\text{LF}_P$ signature $\Sigma$, where $P_1, \ldots, P_n$ is the list of the external predicates occurring in it, if the latter are $\text{LF}$-encoded by $\text{LF}(P_i)$ respectively, calling $\Sigma'$ the $\text{LF}$-signature and $\Gamma'$ the typing context obtained by

- adding to $\Sigma$ $\text{LF}(P_1), \ldots, \text{LF}(P_n)$;
- substituting occurrences of $\mathcal{L}^P_{N,\sigma}[\tau]$ by $(\text{LF}(P) N \sigma) \rightarrow \tau$;

then we have that for each $\text{LF}_P$-derivation $\Gamma \vdash_\Sigma M : \sigma$ where there are no locks neither in $M$ nor in $\sigma$, there is a corresponding $\text{LF}$-derivation $\Gamma' \vdash_{\Sigma'} M' : \sigma'$. 
The “depth” of shallowness (III)

- This correspondence relies on the possibility of representing in LF the external predicates used in LF\(\mathcal{P}\).
- Notice that also \(M\) and \(\sigma\), above, may undergo some changes in the “translation” carried out in LF.
- 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 we presented before;
  - The “deep” one is obtained by implementing the external predicates over the \textit{reification} at object level of the proof terms which were left to the metalanguage in LF\(\mathcal{P}\).
  - The latter encoding is essentially the one of [?] for linear logic.
- 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 \textit{compositional}.
A minimal *expressivity* requirement for a Logical Framework is to be able to represent itself, see e.g. [?, ?].

**Encoding LFₚ within LFₚ:**

- **Basic types:**
  - kind: Type, tp:Type, term: Type, o:Type

- **Term and type constructors:**
  - 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
We do not need to represent explicitly typing environments and signatures: we simply “record” the typings by means of two bookkeeping judgments:

\[
\begin{align*}
tp\text{-}\text{typing}: & \quad \text{tp} \rightarrow \text{kind} \rightarrow \text{Type} \\
typing: & \quad \text{term} \rightarrow \text{tp} \rightarrow \text{Type}
\end{align*}
\]

Encodings of \((O\cdot\text{Lock})\), \((O\cdot\text{Unlock})\) and of the Lock-reduction rule:

\[
\begin{align*}
\text{OLock}: & \quad \Pi M,N:\text{term}. \Pi \rho,\sigma:\text{tp}. \Pi \mathcal{P}:\text{term} \rightarrow \text{o}.
\end{align*}
\]
\[
\begin{align*}
& \quad \text{typing } M \rho \rightarrow \text{typing } N \sigma \rightarrow \\
& \quad \text{typing } (\text{Lock } \mathcal{P} M N \sigma) \text{ (tp}\_\text{Lock } \mathcal{P} \rho N \sigma) \\
\end{align*}
\]

\[
\begin{align*}
\text{OUUnlock}: & \quad \Pi M,N:\text{term}. \Pi \rho,\sigma:\text{tp}. \Pi \mathcal{P}:\text{term} \rightarrow \text{o}.
\end{align*}
\]
\[
\begin{align*}
& \quad \text{typing } M \text{ (tp}\_\text{Lock } \mathcal{P} \rho N \sigma) \rightarrow \text{typing } N \sigma \rightarrow \\
& \quad \mathcal{L} \text{isTrue}_{\langle \mathcal{P},N \rangle} [(\text{typing } (\text{Unlock } \mathcal{P} M N \sigma) \rho)] \\
\end{align*}
\]

\[
\begin{align*}
\text{UL}: & \quad \Pi M,N:\text{term}. \Pi \mathcal{P}:\text{term} \rightarrow \text{o}. \Pi \sigma:\text{tp}.
\end{align*}
\]
\[
\begin{align*}
& \quad \mathcal{L} \text{isTrue}_{\langle \mathcal{P},N \rangle} [(\text{red } (\text{Unlock } \mathcal{P} (\text{Lock } M N \sigma) N \sigma) M)]
\end{align*}
\]

where \(\text{red}: \quad \text{term} \rightarrow \text{term} \rightarrow \text{Type}\) denotes the \(\beta\mathcal{L}\)-reduction judgment on terms and the external predicate \(\mathcal{L}\text{isTrue}\) holds on the pair \(\langle \mathcal{P},N \rangle\) iff predicate \(\mathcal{P}\) holds on \(N\).
The encoding of the typing judgment on terms and types is adequate in the usual sense given by the following theorem:

**Theorem (Adequacy of typing)**

Given $\mathcal{X} = \{x_1, \ldots, x_n\}$ be the set of free variables occurring in $M$ and $\sigma$ 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 $\text{LF}_P$ and proof terms $\mathfrak{h}$, such that $\Gamma' \vdash^n \mathfrak{h} : (\text{typing } \epsilon_{\mathcal{X}}(M) \epsilon_{\mathcal{X}}(\sigma))$ is in canonical form (where $\Gamma' = \{x_1 : \text{term}, \ldots, x_n : \text{term}, h_1 : (\text{typing } x_1 \epsilon_{\mathcal{X}}(\sigma_1)), \ldots, h_n : (\text{typing } x_n \epsilon_{\mathcal{X}}(\sigma_n))\}$).

- where $\epsilon_{\mathcal{X}}$ stands for the encoding function mapping terms and types of $\text{LF}_P$ with free variables in $\mathcal{X}$ into the corresponding canonical forms of type $\text{term}$ and $\text{tp}$, respectively.
- Similar statements can be proved for judgments $\text{red}$ and $\text{tp-typing}$.
We have the following result:

**Theorem**

\( \text{LF}_P \) is a conservative extension of LF.

**Proof (sketch):**

- consider a derivation in \( \text{LF}_P \) and drop all occurrences of locks and unlocks;
- so doing, we obtain a legal derivation in standard LF.

- \( \text{LF}_P \) 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 \( \text{LF}_P \)).
Comparing LFₚ and LF (II)

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 LFₚ into LF:

\[ \text{Unlock: } \Pi M,N: \text{term. } \Pi \rho,\sigma: \text{tp. } \Pi P: \text{term } \rightarrow o. \]

\[ (\text{typing } M (\text{tp Lock } P \rho N \sigma)) \rightarrow (\text{typing } N \sigma) \rightarrow (\text{isTrue } P N) \rightarrow (\text{typing } (\text{Unlock } P M N \sigma) \rho) \]

\[ \text{UL: } \Pi M,N: \text{term. } \Pi P: \text{term } \rightarrow o. \Pi \sigma: \text{tp.} \]

\[ (\text{isTrue } P N) \rightarrow \]

\[ (\text{red } (\text{Unlock } P (\text{Lock } P M N \sigma) N \sigma) M) \]

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 Expressive Power of $\text{LF}_P$

- $\text{LF}_P$ is **decidable**, provided the predicates are decidable;
- Predicates **encoded via inhabitability of a suitable LF type**, as in e.g. AHMP-98 or Crary:2010 for Modal Logic, are well-behaved. In fact, they provide **LF internalizations** of the external predicates used in the $\text{LF}_P$ encodings presented earlier. There is a mapping between signatures and judgements from $\text{LF}_P$ to LF.
- Well-behaved r.e. predicates are LF-definable by Church’s thesis. But how **deep** is the encoding? $\text{LF}_P$ encodings are **shallow** by definition, while encodings via Church’s thesis are deep. *E.g.* “$M, N$ are two different closed normal forms” can be immediately expressed in $\text{LF}_P$.
- $\text{LF}_P$ is a **conservative extension** of LF. Pragmatically, it is very different.
The expressive power of $\lambda\Pi$-calculus modulo is unlimited 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 $\lambda$-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 $\lambda\cap$ [?], which is usually presented only à la Curry.

- In [?], an encoding of the classical PTSs of the $\lambda$-cube, and in particular of the Calculus of Constructions, is given, which can be extended to all GTS’s [?].
LF\_P \ vis-à-vis \ \lambda\Pi\text{-calculus modulo (II)}

- LF\_P can only freeze types, possibly releasing them syntactically unchanged, under suitable circumstances.
- To encode LF\_P in \lambda\Pi\text{-calculus modulo we can reify} the semantics of the lock-operator at the level of types by introducing in Σ:
  - a type \textit{Pred},
  - suitable constants of that type to represent the external predicate,
  - a type constructor

\begin{align*}
\text{lockT} : \Pi P:\textit{Pred}.\Pi \sigma:\textit{Type}.\Pi N: \sigma. \Pi \tau:\textit{Type}.\textit{Type}
\end{align*}

subject to the rewriting rule:

\begin{align*}
(l\text{ockT} \ P \ \sigma \ N \ \tau) \rightarrow^{\Gamma,\textit{Type}} \tau
\end{align*}

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

\[
\begin{align*}
\text{raw\_term} & : \text{Type} \\
\text{type,kind} & : \text{raw\_term} \\
\text{pi,lam} & : \text{raw\_term} \rightarrow (\text{raw\_term} \rightarrow \text{raw\_term}) \rightarrow \text{raw\_term} \\
\text{app} & : \text{raw\_term} \rightarrow \text{raw\_term} \rightarrow \text{raw\_term} \\
\text{typing} & : \text{raw\_term} \rightarrow \text{raw\_term} \rightarrow \text{Type}
\end{align*}
\]

All typing rules of λΠ-calculus modulo can then be represented as suitable axioms involving the typing judgment.

The conversion rule (Conversion \(A \equiv_{\beta R} B\)) of [?] is rendered as:

\[
\begin{align*}
\text{conv} : \Pi A,B,t:\text{raw\_term}. & \quad (\text{typing } A \text{ type}) \rightarrow (\text{typing } B \text{ type}) \rightarrow (\text{typing } t \text{ A}) \rightarrow \\
& \quad L_{\langle A,B \rangle}[(\text{typing } t \text{ B})]
\end{align*}
\]

where the external predicate \(\equiv_{\beta R}\) checks if A and B represent the encodings of two types A and B such that \(A \equiv_{\beta R} B\) holds, according to the rewriting rules.
Comparisons with other systems

- **Conditional LF**: LF\(_P\) is a direct descendant of this system \([?]\):
  - In the latter, the mechanism of freezing a type was achieved by not allowing immediate substitution in the application rule;
  - 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.

- **Pattern LF** \([?]\): it is related both to GLF \([?]\) as well as the rewriting calculi of the \(\rho\)-cube \([?, ?]\):
  - it supports pattern-matching application;
  - thus, it allows both for some freezing of types, as LF\(_P\), and to shortcut the proofs of correctness of computations, in line with Poincaré’s Principle \([?]\) and rewrite-based LFs \([?]\).

- **Linear LF and Concurrent LF**: more work needs to be carried out in order to assess encodings of LLF \([?]\) and CLF \([?]\) in LF\(_P\). What needs to be checked (linearity, order, etc.) can be effectively achieved at abstraction time, e.g., is it ultimately a local property?
Conclusions & Future Work

- $\text{LF}_P$ is an extension of the Edinburgh LF, which internalizes external oracles in the form of a $\diamond$-modal type constructor.
- The main feature of $\text{LF}_P$ is to explicitly separate the true logical contents of derivations from the mere effective verification of other, e.g. syntactical or structural properties. Thus factoring out the complexity of encoding logical systems which are awkward in LF.
- Externally verified predicates provide smoother encodings. The signature is not cluttered by auxiliary notions and mechanisms needed to implement the predicate. This allows for performance optimization.
- The machinery of lock derivations is similar to $\delta$-rules à la Mitschke.
- $\text{LF}_P$ supports in a natural way the “design by contract” programming paradigm:
  - predecessor function on natural numbers: $\lambda x: \text{nat}. \mathcal{L}_{x, \text{nat}}^{>0}[x - 1]$;
  - pre- and post-conditions on a computation $FM$: $\mathcal{L}_M^{P}[\mathcal{L}_M^{Q}[(FM)]]$.
- We are currently planning the development of a prototype implementation of $\text{LF}_P$. 
The traditional LF answer to the question “What is a Logic?” was: “A signature in LF”. In LF$_P$, we can give the homologous answer, namely “A signature in LF$_P$”, since external predicates can be read off the types occurring in the signatures themselves.

But we can also use this very definition to answer a far more intriguing question: “What is a Proof culture?”