$LF_{\mathcal{P}}$ – A Logical Framework with External Predicates

Petar Maksimović

in collaboration with

Furio Honsell, Marina Lenisa, Ivan Scagnetto, and Luigi Liquori

Mathematical Institute of the Serbian Academy of Sciences and Arts, Serbia
Faculty of Technical Sciences, University of Novi Sad, Serbia
INRIA Sophia Antipolis Méditerranée, France
Universitá di Udine, Italy

Logic and Applications 2013, September 16-20, 2013,
Dubrovnik, Croatia

Table of Contents

Logical Frameworks

- Formal systems based on a typed λ-calculus
- Connected (somewhat unexpectedly) with proof systems via the Curry-Howard Correspondence, interpreting formulas-as-types and proofs-as-programs.
- Serve as bases for various interactive theorem provers
  - Coq, Lego, Twelf
- Harper-Honsell-Plotkin's Edinburgh Logical Framework - LF
  - Featuring dependent types - types depending on terms
- Coquand's Calculus of Constructions
  - Featuring type polymorphism, dependent types, and higher-order types.

What would we like to do?

- Difficult and cumbersome encodings of side conditions - we would like to make that a little easier and a lot more natural.
- Somehow separate derivation and computation - maybe have conditions verified externally.
- Allow the interaction and co-operation of various formal provers. Maybe even leave room for some "informal" mechanisms.

Pseudo-syntax of $LF_{\mathcal{P}}$

- Five syntactic categories: signatures (for type and term constants), contexts (for variables), kinds, types, and terms.

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

Judgements in $\mathrm{LF}_{\mathcal{P}}$

The type system for $\mathrm{LF}_{\mathcal{P}}$ proves the following five judgements:

Σ sig	Σ is a valid signature
$\vdash_\Sigma \Gamma$	Γ is a valid context in Σ
$\Gamma \vdash_\Sigma K$	K is a kind in Γ and Σ
$\Gamma \vdash_\Sigma \sigma : K$	σ has kind K in Γ and Σ
$\Gamma \vdash_\Sigma M : \sigma$	M has type σ in Γ and Σ

Type System for $LF_{\mathcal{P}}$ - Signatures

Signatures serve to keep track of constant types and terms.

- An empty signature is a valid signature.

$$\frac{}{\emptyset \ \mathsf{sig}}$$

- A valid signature Σ can be extended with a fresh family a, whose kind K is a kind in the empty context and signature Σ.

$$\frac{\Sigma \ \mathsf{sig} \quad \vdash_\Sigma K \quad a \notin \mathrm{Dom}(\Sigma)}{\Sigma, a{:}K \ \mathsf{sig}}$$

- A valid signature Σ can be extended with a fresh object c, whose type σ has kind Type in the empty context and signature Σ.

$$\frac{\Sigma \ \mathsf{sig} \quad \vdash_\Sigma \sigma{:}\mathrm{Type} \quad c \notin \mathrm{Dom}(\Sigma)}{\Sigma, c{:}\sigma \ \mathsf{sig}}$$

Type System for $LF_{\mathcal{P}}$ - Contexts

Contexts keep track of variables.

- An empty context is a valid context in any valid signature Σ.

$$\frac{\Sigma \ \mathrm{sig}}{\vdash_\Sigma \emptyset}$$

- A valid context Γ in the signature Σ can be extended with a fresh variable x, whose type is of kind Type in Γ and Σ.

$$\frac{\vdash_\Sigma \Gamma \quad \Gamma \vdash_\Sigma \sigma{:}\mathrm{Type} \quad x \notin \mathrm{Dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}\sigma}$$

Type System for $\text{LF}_\mathcal{P}$ - Kinds

- If Γ is a valid context in the signature Σ, then Type is a kind in Γ and Σ.

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \text{Type}}$$

- If K is a kind in the context $\Gamma, x{:}\sigma$ and signature Σ, then the dependent product $\Pi x{:}\sigma.K$ is a kind in Γ and Σ.

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

Type System for $LF_{\mathcal{P}}$ - Types

- If Γ is a valid context in the signature Σ, then any family a of kind K belonging to Σ also has kind K in Γ and Σ.

$$\frac{\vdash_{\Sigma} \Gamma \quad a{:}K \in \Sigma}{\Gamma \vdash_{\Sigma} a : K}$$

- If τ has kind Type in the context $\Gamma, x{:}\sigma$ and signature Σ, then the dependent product $\Pi x{:}\sigma.\tau$ has kind Type in Γ and Σ.

$$\frac{\Gamma, x{:}\sigma \vdash_{\Sigma} \tau : \mathrm{Type}}{\Gamma \vdash_{\Sigma} \Pi x{:}\sigma.\tau : \mathrm{Type}}$$

- If σ has kind $\Pi x{:}\tau.K$ in the context Γ and signature Σ, and N has type τ in Γ and Σ, then the application of N to σ has kind K, in which all occurrences of x have been substituted for N, in Γ and Σ.

$$\frac{\Gamma \vdash_{\Sigma} \sigma : \Pi x{:}\tau.K \quad \Gamma \vdash_{\Sigma} N : \tau}{\sigma\, N : K[N/x]}$$

Type System for $LF_{\mathcal{P}}$ - More Types

- If ρ has kind K in the context Γ and signature Σ, and N has type σ in Γ and Σ, then the type locking ρ with a predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N : \sigma$ has kind Type in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma \rho : \text{Type} \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \text{Type}}$$

- If σ has kind K in the context Γ and signature Σ, and K is definitionally equal to K', which is a kind in Γ and Σ, then σ also has kind K' in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma \sigma : K \quad \Gamma \vdash_\Sigma K' \quad K =_{\beta\mathcal{L}} K'}{\Gamma \vdash_\Sigma \sigma : K'}$$

Type System for $LF_{\mathcal{P}}$ - Terms

- If Γ is a valid context in the signature Σ, then any object c of type σ belonging to Σ also has type σ in Γ and Σ.

$$\frac{\vdash_\Sigma \Gamma \quad c{:}\sigma \in \Sigma}{\Gamma \vdash_\Sigma c : \sigma}$$

- If Γ is a valid context in the signature Σ, then any variable x of type σ belonging to Γ also has type σ in Γ and Σ.

$$\frac{\vdash_\Sigma \Gamma \quad x{:}\sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma}$$

- If M has type τ in the context $\Gamma, x{:}\sigma$ and signature Σ, then the abstraction $\lambda x{:}\sigma.M$ has type $\Pi x{:}\sigma.\tau$ in Γ and Σ.

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma M : \tau}{\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau}$$

Type System for $LF_{\mathcal{P}}$ - More Terms

- If M has type $\Pi x{:}\sigma.\tau$ in the context Γ and signature Σ, and N has type σ in Γ and Σ, then the application of N to M has type τ, in which all occurrences of x have been substituted for N, in Γ and Σ.

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

- If M has type ρ in the context Γ and signature Σ, and N has type σ in Γ and Σ, then M, locked with the predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N : \sigma$ has type ρ, locked with the predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N : \sigma$, in Γ and Σ.

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

Type System for $\mathrm{LF}_{\mathcal{P}}$ - Even More Terms

- If M has type ρ, locked with the predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N{:}\sigma$ in the context Γ and signature Σ, N has type σ in Γ and Σ, and $\mathcal{P}(\Gamma \vdash_\Sigma N{:}\sigma)$ holds, then M, unlocked with $\mathcal{P}$ on $\Gamma \vdash_\Sigma N{:}\sigma$ has type ρ in Γ and Σ.

$$\frac{\begin{array}{l}\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)\\ \Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]\end{array} \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho}$$

- If M has type σ in the context Γ and signature Σ, and σ is definitionally equal to σ', which has kind Type in Γ and Σ, then M also has type σ' in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma M : \sigma \quad \Gamma \vdash_\Sigma \sigma' : \text{Type} \quad \sigma =_{\beta\mathcal{L}} \sigma'}{\Gamma \vdash_\Sigma M : \sigma'}$$

Definitional Equality in $\mathsf{LF}_{\mathcal{P}}$

In $\mathsf{LF}_{\mathcal{P}}$, there are two types of reduction:

- Standard β-reduction on the level of kinds, types, and terms:

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

- A new form of reduction, $\mathcal{L}$-reduction, on the level of terms, where a lock dissolves in the presence of an unlock:

$$\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} M.$$

Notice that predicate validity check is required for the unlock constructor to be applied, and not during reduction. Also, there is no need for $\mathcal{L}$-reduction at the level of types.

Strong Normalization

- We will rely on the Strong Normalization of LF.
- Let us begin by defining the function $^{-\mathcal{UL}}$: $LF_{\mathcal{P}} \to LF$:
  1. $\text{Type}^{-\mathcal{UL}} = \text{Type}$, $a^{-\mathcal{UL}} = a$, $c^{-\mathcal{UL}} = c$, $x^{-\mathcal{UL}} = x$,
  2. $(\Pi x{:}\sigma.T)^{-\mathcal{UL}} = \Pi x{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}}$,
  3. $(\lambda x{:}\sigma.T)^{-\mathcal{UL}} = \lambda x{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}}$,
  4. $(T\,M)^{-\mathcal{UL}} = T^{-\mathcal{UL}}\,M^{-\mathcal{UL}}$,
  5. $(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[T])^{-\mathcal{UL}} = (\lambda x_f{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}})\,N^{-\mathcal{UL}}$,
  6. $(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[T])^{-\mathcal{UL}} = (\lambda x_f{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}})\,N^{-\mathcal{UL}}$,

  which maps derivable judgements of $LF_{\mathcal{P}}$ into derivable judgements of LF, stripping away the $\mathcal{L}$ and $\mathcal{U}$.
- Note the free variable x_f, which preserves N and σ from the lock and unlock operators.

Strong Normalization

- $\mathcal{L}$-reductions cannot create new β-redexes in T, but can only "unlock" them, and these unlocked redexes remain in $T^{-\mathcal{UL}}$:

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

- Therefore, at least as many β-reductions can be performed in $T^{-\mathcal{UL}}$ as can be performed in T:

$$\max_{\beta}(T) \leq \max_{\beta}(T^{-\mathcal{UL}}) < \infty.$$

- There is no $LF_{\mathcal{P}}$ term T with an infinite β-reduction sequence.

Strong Normalization

Theorem (Strong normalization of $LF_{\mathcal{P}}$)

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

Proof for all three cases.

Let us suppose that $\max_{\beta\mathcal{L}}(T) = \infty$. Then, it must be that $\max_{\mathcal{L}}(T) = \infty$. But, we initially only have finitely many $\mathcal{L}$-redexes, and this can increase only by a finite number at a time (through β-reduction). Therefore, it must be that $\max_{\beta}(T) = \infty$, which is not possible. □

Confluence

- First, we prove the following lemma:

Lemma (Local confluence of $\mathrm{LF}_{\mathcal{P}}$)

$\beta\mathcal{L}$-reduction is locally confluent, i.e.

- *If $T \to_{\beta\mathcal{L}} T'$ and $T \to_{\beta\mathcal{L}} T''$, then there exists a T''', such that $T' \twoheadrightarrow_{\beta\mathcal{L}} K'''$ and $T'' \twoheadrightarrow_{\beta\mathcal{L}} T''$.*

- Then, using *Newman's lemma* (local confluence + strong normalization $\to$ confluence), we obtain:

Theorem (Confluence of $\mathrm{LF}_{\mathcal{P}}$)

$\beta\mathcal{L}$-reduction is confluent, i.e.

- *If $T \twoheadrightarrow_{\beta\mathcal{L}} T'$ and $T \twoheadrightarrow_{\beta\mathcal{L}} T''$, then there exists a T''', such that $T' \twoheadrightarrow_{\beta\mathcal{L}} K'''$ and $T'' \twoheadrightarrow_{\beta\mathcal{L}} T''$.*

Subject Reduction

This time, we need additional conditions on the predicates:

Definition (Well-behaved predicates)

A predicate $\mathcal{P}$ is *well-behaved* if it satisfies the following conditions:

Closure under signature and context weakening and permutation:

- $\Sigma, \Omega \text{ sig}, \Sigma \subseteq \Omega, \mathcal{P}(\Gamma \vdash_\Sigma \alpha) \rightarrow \mathcal{P}(\Gamma \vdash_\Omega \alpha)$.
- $\vdash_\Sigma \Gamma, \vdash_\Sigma \Delta, \Gamma \subseteq \Delta, \mathcal{P}(\Gamma \vdash_\Sigma \alpha) \rightarrow \mathcal{P}(\Delta \vdash_\Sigma \alpha)$.

Closure under substitution:

- $\mathcal{P}(\Gamma, x{:}\sigma', \Gamma' \vdash_\Sigma N : \sigma), \Gamma \vdash_\Sigma N' : \sigma' \rightarrow$ $\mathcal{P}(\Gamma, \Gamma'[N'/x] \vdash_\Sigma N[N'/x] : \sigma[N'/x])$.

Closure under reduction:

- $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma), N \rightarrow_{\beta\mathcal{L}} N' \rightarrow \mathcal{P}(\Gamma \vdash_\Sigma N' : \sigma)$.
- $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma), \sigma \rightarrow_{\beta\mathcal{L}} \sigma' \rightarrow \mathcal{P}(\Gamma \vdash_\Sigma N : \sigma')$.

Subject Reduction

With the well-behavedness conditions imposed on predicates, and several more standard auxiliary lemmas, including:

- subderivation,
- weakening and permutation,
- transitivity,
- unicity of types and kinds,

we can prove subject reduction of $LF_{\mathcal{P}}$:

Theorem (Subject reduction of $LF_{\mathcal{P}}$)

If predicates are well-behaved, then:

1. *If* $\Gamma \vdash_\Sigma K$, *and* $K \to_{\beta\mathcal{L}} K'$, *then* $\Gamma \vdash_\Sigma K'$.
2. *If* $\Gamma \vdash_\Sigma \sigma : K$, *and* $\sigma \to_{\beta\mathcal{L}} \sigma'$, *then* $\Gamma \vdash_\Sigma \sigma' : K$.
3. *If* $\Gamma \vdash_\Sigma M : \sigma$, *and* $M \to_{\beta\mathcal{L}} M'$, *then* $\Gamma \vdash_\Sigma M' : \sigma$.

Other Properties of $LF_{\mathcal{P}}$

- $LF_{\mathcal{P}}$ is decidable, if the predicates are decidable.
- If a predicate is *definable in* LF, *i.e.* if it can be encoded via the inhabitability of a suitable LF dependent type, then it is well-behaved.
- All well-behaved recursively enumerable predicates are LF-definable by Church's thesis. But not that easily. Consider *e.g.* the well-behaved predicate "M, N are two different closed normal forms", which can be immediately expressed in $LF_{\mathcal{P}}$.

η-long Normal Forms in $LF_{\mathcal{P}}$

Definition (Fully applied and unlocked occurrences)

An occurrence ξ of a constant or a variable in a term of an $LF_{\mathcal{P}}$ judgement is *fully applied and unlocked* with respect to its type or kind $\Pi\vec{x}_1{:}\vec{\sigma}_1.\vec{\mathcal{L}}_1[\ldots\Pi\vec{x}_n{:}\vec{\sigma}_n.\vec{\mathcal{L}}_n[\alpha]\ldots]$, where $\vec{\mathcal{L}}_1,\ldots,\vec{\mathcal{L}}_n$ are vectors of locks, if ξ appears in contexts of the form $\vec{\mathcal{U}}_n[(\ldots(\vec{\mathcal{U}}_1[\xi\vec{M}_1])\ldots)\vec{M}_n]$, where $\vec{M}_1,\ldots,\vec{M}_n$, $\vec{\mathcal{U}}_1,\ldots,\vec{\mathcal{U}}_n$ have the same arities of the corresponding vectors of Π's and locks.

Definition (Judgements in η-long normal form)

- A term T in a judgement is in *η-lnf* if T is in normal form and every constant and variable occurrence in T is fully applied and unlocked w.r.t. its classifier in the judgement.
- A judgement is in *η-lnf* if all terms appearing in it are in η-lnf.

Untyped λ-calculus

- The syntax:

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

- The strategy:
  - Higher-Order-Abstract-Syntax (HOAS)
  - Delegating α-conversion and capture-avoiding substitution to the metalanguage.
  - Modeling free and bound variables so that the well-behavedness conditions for the predicates are met.

- Signature Σ_λ for the untyped λ-calculus in $LF_{\mathcal{P}}$:

`nat : Type`	The type of natural numbers
`0 : nat`	Zero is a natural number
`S : nat -> nat`	The successor function
`free : nat -> term`	Modeling free variables
`app : term -> term -> term`	Application
`lam : (term -> term) -> term`	Abstraction

Untyped λ-calculus

- Natural numbers encoded in the standard way
- Variables of the untyped λ-calculus enumerated: $\{x_i\}_{i \in \mathbb{N} \setminus \{0\}}$
- The encoding function $\epsilon_{\mathcal{X}}$, mapping the terms of the untyped λ-calculus into terms of $LF_{\mathcal{P}}$:

$$\epsilon_{\mathcal{X}}(x_i) = \begin{cases} \mathtt{xi}, \text{ if } x_i \in \mathcal{X} \\ (\mathtt{free\ i}), \text{ if } x_i \notin \mathcal{X} \end{cases},$$

$$\epsilon_{\mathcal{X}}(MN) = (\mathtt{app}\ \epsilon_{\mathcal{X}}(M)\, \epsilon_{\mathcal{X}}(N)),$$

$$\epsilon_{\mathcal{X}}(\lambda x_i.M) = (\mathtt{lam}\ \lambda \mathtt{x_i}{:}\mathtt{term}.\epsilon_{\mathcal{X} \cup \{\mathtt{x_i}\}}(M)).$$

- Therefore, $\epsilon_{\emptyset}(x_n) = (\mathtt{free\ n})$, but

$$\epsilon_{\emptyset}(\lambda x_n.x_n) = (\mathtt{lam}\ \lambda \mathtt{x_n}{:}\mathtt{term}.\epsilon_{\{\mathtt{x_n}\}}(\mathtt{x_n})) = (\mathtt{lam}\ \lambda \mathtt{x_n}{:}\mathtt{term}.\mathtt{x_n}).$$

Untyped λ-calculus

- In this way, we ensure that we can abide by the "closure under substitution" condition for the predicates, while still retaining the ability to handle "open" terms explicitly.
- We have the following adequacy theorem:

Theorem (Adequacy of syntax)

Let $\{x_i\}_{i \in \mathbb{N} \setminus \{0\}}$ be an enumeration of the variables in the λ-calculus. Then, the encoding function $\epsilon_{\mathcal{X}}$ is a bijection between the λ-calculus terms with bindable variables in $\mathcal{X}$ and the terms M derivable in judgements $\Gamma \vdash_{\Sigma_\lambda} M : \mathtt{term}$ in η-lnf, where $\Gamma = \{\mathtt{x} : \mathtt{term} \mid x \in \mathcal{X}\}$.

- However, here we don't use the main features of $LF_{\mathcal{P}}$ - locked types and external predicates. So, let us try to add to this encoding a call-by-value reduction strategy.

Untyped λ-calculus with call-by-value reduction

- Reduction induces an equivalence relation on the set of terms:

```
eq : term -> term -> Type
```

- Symmetry:

$$\frac{\vdash_{CBV} N = M}{\vdash_{CBV} M = N}$$

```
symm : ΠM:term.ΠN:term.(eq N M) -> (eq M N)
```

- Conditional β-reduction:

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

$$\texttt{betav : } \Pi\texttt{M:(term -> term).}\Pi\texttt{N:term.}$$
$$\mathcal{L}^{\texttt{Val}}_{\texttt{N,term}}[\texttt{eq (app (lam M) N) (M N)}]$$

Untyped λ-calculus with call-by-value reduction

- Conditional β-reduction:

$$\texttt{betav} : \Pi\texttt{M}:(\texttt{term -> term}).\Pi\texttt{N}:\texttt{term}.$$
$$\mathcal{L}^{\mathit{Val}}_{\texttt{N,term}}[\texttt{eq (app (lam M) N) (M N)}]$$

- The predicate $\mathit{Val}(\Gamma \vdash_\Sigma \texttt{N} : \texttt{term})$ holds iff either N is an abstraction or a constant (a term of the shape (free i));

$\mathit{Val}(\Gamma \vdash_\Sigma \texttt{N} : \texttt{term}) \Rightarrow$

```
let norm=NF(N) in
  match norm with
    | app M' N' => false
    | _ => true
  end
```

- The predicate Val is well-behaved.

Untyped λ-calculus with call-by-value reduction

Theorem (Adequacy of CBV reduction)

Given an enumeration $\{x_i\}_{i \in \mathbb{N} \setminus \{0\}}$ *of the variables in the* λ*-calculus, there is a bijection between derivations of the judgement* $\vdash_{CBV} M = N$ *on terms with no bindable variables in the CBV* λ*-calculus and proof terms* $\mathtt{h}$*, such that* $\vdash_{\Sigma_{CBV}} \mathtt{h} : (\mathtt{eq}\ \epsilon_{\emptyset}(M)\ \epsilon_{\emptyset}(N))$ *is in* η*-lnf.*

Necessitation in Modal Logics

- Side-conditions on application of inference rules:

From ϕ infer $\Box\phi$, if ϕ is a theorem.

- We can encode this in $LF_{\mathcal{P}}$ with relative ease:

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

where `o : Type` is the type of propositions, and `True : o -> Type` is the truth judgement.

- The predicate $\mathit{Closed}(\Gamma \vdash_\Sigma \mathtt{m}{:}\mathit{True}(\phi))$ holds iff "all of the free variables that occur in `m` are of type `o`".
- The predicate inspects the environment and has to be defined on *typed judgements*.

Several further examples

- **Capturing π-calculus.** The reduction rule taking into account structural congruences between processes, namely

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

can be easily encoded in $LF_{\mathcal{P}}$ as:

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

where `red` encodes the reduction relation $\longrightarrow$, and the *external* predicate `Struct` holds iff $P \equiv P'$ and $Q' \equiv Q$.

- **Capturing Deduction Modulo.** The rule:

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

can be encoded as:

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

What did we get with $LF_{\mathcal{P}}$?

- A mechanism allowing the interconnection of different formal (and informal) verification tools.
- Easy encodings of side-conditions on applications of rules.
- Subsumption of a number of well-known formal systems from the literature.
- An elegant separation between derivation and computation.
- Cleaner and more readable proofs.

The end of the presentation

Thank you for your attention!

Any questions?