3^{rd} Workshop on the Rewriting Calculus

London, Kings College, October 23-24, 2006

A Framework for Defining Logical Frameworks

Luigi Liquori (INRIA Sophia-Antipolis, France)

joint work with

Furio Honsell (Università di Udine, Italia)
Marina Lenisa (Università di Udine, Italia)

dedicated to Gordon D. Plotkin,
on the occasion of his 60^{th} *birthday*

The Problem's Setting

- 40 years on, it is unchallenged that: **rigorous certification** is necessary to improve **software certification**, and this calls for the use of **Formal Methods** and hence **Formal Systems**

- **Logic** as **Applied Mathematics**

BUT

- The **Ultimate Formal System**, *i.e.* the system which provides **in all situations** transparent encodings, thus allowing to reason both naturally and formally, **does not** appear to **exist**

- We live in a world where new Formal Systems are constantly emerging and often are quickly forgotten before they are fully understood - nevertheless the approach works, at least **locally**!

- **Logics** as **Applied Mathematics**

Systems

- Separation Logics
- Temporal Logics
- Digraphs for modeling space and time
- Logics for artificial biochemistry
- Logics for global and concurrent computing
- Logics for access control
- Logics for data update
- ...

The Problem

- There do exist concepts which are **sensitive to encoding** in the metalanguage
- In order to carry out complex reasoning, we need to move from one Formal System to another
- The **Ultimate Formal System (UFS)** might not exist, but we can hope that, at the metalevel, the **Ultimate Logical Framework (ULF)** might exist
- Does UFS exist? if not, does the ULF exist?
- **FOL** could do, but it **does** provide a **heavy encoding overhead**

A little History - I

- Exactly 20 years on Gordon Plotkin, was experimenting with **Typed λ-calculi** and **Curry-Howard's Proposition-as-types paradigm** in order to build a **general logic**

- Proposition-as-types = typed λ-terms correspond to the (constructive) proofs of the proposition represented by the type

And a number of things fell into place in the Summer and Fall of 1986

- Martin-Löf introduced the notion of Judgment, Coquand and Huet the Calculus of Constructions, Meyer and Rheingold the Π-calculus

- The **Laboratory for the Foundations of Computer Science LFCS** was founded in Edinburgh - many RA's were hired, including Bob Harper and Furio Honsell - to work on a **Generic Proof Editor**

- Generic proof editor = should **factor out** as much as possible, so that one should **not restart form scratch** every time one wanted to build a semi-automatic proof assistant for a given logic. The template was the **LCF** environment

A little History - II

"A Framework for Defining Logics" by Harper, Honsell, Plotkin, submitted December 1986 to LICS (published in JACM in 1993) put forward

- **LF**, a simple dependent typed λ-calculus, as a **Generic Logic of Judgments**, and hence as a **Logic Specification Language**
  - the **higher order** nature of the system provides a natural setting for representing **reasoning under assumptions**, *i.e.* logics in **natural deduction** style, or **using lemma**
  - **higher order dependent** types naturally enforce side conditions related to **generality** and **locality** of free variables
  - hence types represent **hypothetico general** Judgments
  - object language **variables** and **substitution** are modeled by the corresponding meta-language notions
- the **Judgments-as-Types** paradigm *i.e.* a Judgment is **provable** if and only if the corresponding type is **inhabited**

The LF paradigm for encoding a Logic $\mathcal{L}$

- **LF types** encode the **syntactic categories** as well as the **Judgments** of the logic $\mathcal{L}$
- **Well typed ground LF terms** represent **wff** in the logic as well as **proofs of judgments**
- **Well typed higher order LF terms** represent both **binding operators** and **rule schemata**
- So, **substitution** and **instantiation** are just **λ-application**

The features of LF

- **MORAL: typed λ-calculus naturally captures all notions, machinery and pragmatics of logic specification and interactive proof development**
- **Decidability of type checking** in LF expresses the **decidability of proof correctness**
- LF is **logically normative**
- LF is **self encoding**
- LF provides an excellent analysis of **derivable** and **admissible** rules

Feferman, Constable, Paulson, Pfenning showed that the **LF game** could be played also in their systems viz Recursion Theory, NUPRL, Isabelle, ELF, TWELF, HOL, in as much they were based on λ-calculus. The **Nancy School** plays this game with the **typed ρ-calculus**.

An example: the Signature for Propositional Logic *à la* Hilbert

Propositional Connectives

$o : \mathsf{Type} \qquad \supset : o \to o \to o \qquad \neg : o \to o$

Judgment

$\mathsf{True} : o \to \mathsf{Type}$

Propositional Axioms

$\mathsf{A}_1 \ : \Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}\phi \supset (\psi \supset \phi)$

$\mathsf{A}_2 \ : \Pi\phi^o.\ \Pi\psi^o.\ \Pi\theta^o.\ \mathsf{True}(\phi \supset (\psi \supset \theta)) \supset (\phi \supset \psi) \supset (\phi \supset \theta)$

$\mathsf{A}_3 \ : \Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}(\neg\psi \supset \neg\phi) \supset (\neg\psi \supset \phi) \supset \psi$

Rule

$\mathsf{MP} \ : \Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}\phi \supset \mathsf{True}(\phi \supset \psi) \to \mathsf{True}\psi$

Case studies over the years

- FOL, HOL, Type Theories, various λ-calculi
- Modal Logics
- Program Logics: Hoare's and Dynamic
- π-calculus and Nominal Calculi
- ρ-calculus and Lambda+Rewriting

Critical issues

So what do you do with a system once you have it? If you are a natural philosopher and you love knowledge, you **try to break it**! And go and look for peculiar systems

- Encodings of **Modal Logics** are not so natural after all. Side conditions such as in **rules of proof**, (*i.e.* can be applied only when they depend on no assumptions) or in **rules of derivation**, (*i.e.* can be applied only if **all assumptions are boxed**) can be encoded only indirectly

- Similarly for **Program Logics** which feature certain rules which can be applied **only in special contexts** (*e.g.* at top level)

One would like to be able to express syntactic conditions on terms, such as **being closed**, **not having any variable whose type contains certain constants, linearity, . . .**

20 years later: The General Logical Framework

The idea of a **Framework for Defining Logical Frameworks** grew out of the need

- to extend the class of **syntactical conditions on proof terms** which can be expressed naturally

- To incorporate in LF, **term rewriting systems and pattern-matching features** like in the **Typed Rewriting Calculus** *à la* Cirstea-Kirchner-Liquori (FOSSACS-01, TYPES-03, POPL-03)

- Jouannaud *et al.* share our enthusiasm and goal using a different "rewriting spirit"

$$incorporate \longrightarrow cohabit$$

20 years later: The General Logical Framework GLF

- The solution arrived when we realized that: ***sometimes less is more*** (who said that first?) and we removed a **blind spot** in the LF's type system

$$\frac{\Gamma \vdash M : \Pi x{:}\sigma.\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau[N/x]} \qquad \frac{\Gamma \vdash M : \Pi x{:}\sigma.\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : (\lambda x{:}\sigma.\tau)\ N}$$

- We replaced $\lambda x{:}\sigma.M$ and $\Pi x{:}\sigma.\tau$ by a **new form** of λ and Π abstraction $\lambda\mathcal{P}{:}\Delta.M$

- The predicate $\mathcal{P}$ is **completely general** at this stage, and

$$\Delta \triangleq x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$$

denotes the **bound variables** by Π and λ in $\Pi\mathcal{P}{:}\Delta.\tau$ and $\lambda\mathcal{P}{:}\Delta.M$, and

$(\lambda\mathcal{P}{:}\Delta.M)\ N$ reduces to $M\widehat{\mathcal{P}}(N)$ if $\mathcal{P}(N)$ holds

- $\widehat{\mathcal{P}}(N)$ is a **substitution** on $\mathsf{Dom}(\Delta)$, **otherwise** it gets stuck, until possible further substitutions will make it fire

Hat and Bar (Postpone?)

In order to **instantiate** GLF, we need to specify $\mathcal{P}$. This amounts to defining a **typability precondition** and a **substitution**

1. Let $\bar{}$: $[\mathcal{L} \to [\mathcal{C} \to \mathsf{GLF}]]$ be a function taking a predicate $\mathcal{P}$ and a context Δ, and producing a term whose free variables are in $\mathsf{Dom}(\Delta)$. Informally, $\bar{}$ gives the **term subexpressions of the predicate**. It will be made clear in the typing rules
2. Let $\hat{}$: $[\mathcal{L} \to [\mathcal{C} \to [\mathsf{GLF} \to \mathcal{S}ub]_\perp]]$ be a function taking a $\mathcal{P}$ and a Δ, and producing a partial function that takes a term M and produces a **substitution** over $\mathsf{Dom}(\Delta)$, provided M satisfies $\mathcal{P}$. It will be used in reductions

GLF's Syntax and Judgments

$$\Sigma \quad ::= \quad \varnothing \mid \Sigma, a{:}K \mid \Sigma, f{:}\sigma \qquad \textbf{Signatures}$$

$$\Gamma \quad ::= \quad \varnothing \mid \Gamma, x{:}\sigma \qquad \textbf{Contexts}$$

$$K \quad ::= \quad \mathsf{Type} \mid \Pi\mathcal{P}{:}\Delta.K \mid \lambda\mathcal{P}{:}\Delta.K \mid K\,M \qquad \textbf{Kinds}$$

$$\sigma \quad ::= \quad a \mid \Pi\mathcal{P}{:}\Delta.\sigma \mid \lambda\mathcal{P}{:}\Delta.\sigma \mid \sigma\,M \qquad \textbf{Types (Families)}$$

$$M \quad ::= \quad f \mid x \mid \lambda\mathcal{P}{:}\Delta.M \mid M\,N \qquad \textbf{Terms (Objects)}$$

$$\Sigma\ \mathsf{sig} \qquad \vdash_\Sigma \Gamma \qquad \Gamma \vdash_\Sigma K \qquad \Gamma \vdash \sigma : K \qquad \Gamma \vdash_\Sigma M : \sigma$$

GLF's Reductions (top level)

$$(\beta_{\mathcal{P}}-\mathsf{Terms})\quad (\lambda\mathcal{P}{:}\Delta.M)\,N \rightarrow_{\beta_{\mathcal{P}}} M\,\widehat{\mathcal{P}}(N)$$

$$(\beta_{\mathcal{P}}-\mathsf{Types})\quad (\lambda\mathcal{P}{:}\Delta.\sigma)\,N \rightarrow_{\beta_{\mathcal{P}}} \sigma\,\widehat{\mathcal{P}}(N)$$

$$(\beta_{\mathcal{P}}-\mathsf{Kinds})\quad (\lambda\mathcal{P}{:}\Delta.K)\,N \rightarrow_{\beta_{\mathcal{P}}} K\,\widehat{\mathcal{P}}(N)$$

Let $\widehat{\ }$: $[\mathcal{L} \rightarrow [\mathcal{C} \rightarrow [\mathsf{GLF} \rightarrow \mathcal{S}ub]_{\perp}]]$ be a function taking a $\mathcal{P}$ and a Δ, and producing a partial function that takes a term M and produces a **substitution** over $\mathsf{Dom}(\Delta)$, provided M satisfies $\mathcal{P}$. We denote $\widehat{\mathcal{P}}(\Delta)$ simply by $\widehat{\mathcal{P}}$

Signature and Context Rules

$$\frac{}{\varnothing \text{ sig}}\,(\text{S·Empty}) \qquad \frac{\Sigma \text{ sig} \quad \vdash_\Sigma K \quad a \notin \text{Dom}(\Sigma)}{\Sigma, a{:}K \text{ sig}}\,(\text{S·Kind})$$

$$\frac{\Sigma \text{ sig} \quad \vdash_\Sigma \sigma : \text{Type} \quad f \notin \text{Dom}(\Sigma)}{\Sigma, f{:}\sigma \text{ sig}}\,(\text{S·Type})$$

$$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \varnothing}\,(\text{C·Empty}) \qquad \frac{\vdash_\Sigma \Gamma \quad \Gamma \vdash_\Sigma \sigma : \text{Type} \quad x \notin \text{Dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}\sigma}\,(\text{C·Type})$$

Kind Rules

Let $\overline{\ }: [\mathcal{L} \to [\mathcal{C} \to \mathsf{GLF}]]$ be a function taking a predicate $\mathcal{P}$ and a context Δ, and producing a term whose free variables are in $\mathsf{Dom}(\Delta)$. Informally, $\overline{\ }$ gives the **term subexpressions of the predicate**. We denote $\overline{\mathcal{P}}(\Delta)$ simply by $\overline{\mathcal{P}}$

$$\frac{\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma, \Delta \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi \mathcal{P}{:}\Delta.K}\,(\mathsf{K{\cdot}Pi})$$

$$\frac{\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma, \Delta \vdash_\Sigma K}{\Gamma \vdash_\Sigma \lambda \mathcal{P}{:}\Delta.K}\,(\mathsf{K{\cdot}Abs})$$

$$\frac{\Gamma \vdash_\Sigma \Pi \mathcal{P}{:}\Delta.K \quad \Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \underbrace{(\lambda \mathcal{P}{:}\Delta.K)\, N}_{\text{redex?}}}\,(\mathsf{K{\cdot}Appl})$$

Family Rules

$$\frac{\vdash_\Sigma \Gamma \quad a{:}K \in \Sigma}{\Gamma \vdash_\Sigma a : K}\,(\text{F}\cdot\text{Var})$$

$$\frac{\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma, \Delta \vdash_\Sigma \tau : \text{Type}}{\Gamma \vdash_\Sigma \Pi \mathcal{P}{:}\Delta.\tau : \text{Type}}\,(\text{F}\cdot\text{Pi})$$

$$\frac{\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma, \Delta \vdash_\Sigma \tau : K}{\Gamma \vdash_\Sigma \lambda \mathcal{P}{:}\Delta.\tau : \Pi \mathcal{P}{:}\Delta.K}\,(\text{F}\cdot\text{Abs})$$

$$\frac{\Gamma \vdash_\Sigma \sigma : \Pi \mathcal{P}{:}\Delta.K \quad \Gamma \vdash_\Sigma M : \tau}{\Gamma \vdash_\Sigma \sigma\, M : \underbrace{(\lambda \mathcal{P}{:}\Delta.K)\, M}_{\text{redex?}}}\,(\text{F}\cdot\text{Appl})$$

$$\frac{\Gamma \vdash_\Sigma \sigma : K' \quad \Gamma \vdash_\Sigma K \quad \Gamma \vdash_\Sigma K =_{\beta_\mathcal{P}} K'}{\Gamma \vdash_\Sigma \sigma : K}\,(\text{F}\cdot\text{Conv})$$

Object Rules

$$\frac{\vdash_\Sigma \Gamma \quad f{:}\sigma \in \Sigma}{\Gamma \vdash_\Sigma f : \sigma}\,(\mathsf{O{\cdot}Const}) \qquad \frac{\vdash_\Sigma \Gamma \quad x{:}\sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma}\,(\mathsf{O{\cdot}Var})$$

$$\frac{\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma, \Delta \vdash_\Sigma M : \tau}{\Gamma \vdash_\Sigma \lambda \mathcal{P}{:}\Delta.M : \Pi \mathcal{P}{:}\Delta.\tau}\,(\mathsf{O{\cdot}Abs})$$

$$\frac{\Gamma \vdash_\Sigma M : \Pi \mathcal{P}{:}\Delta.\tau \quad \Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma M\ N : \underbrace{(\lambda \mathcal{P}{:}\Delta.\tau)\ N}_{\text{redex?}}}\,(\mathsf{O{\cdot}Appl})$$

$$\frac{\Gamma \vdash_\Sigma M : \sigma \quad \Gamma \vdash_\Sigma \tau : \mathsf{Type} \quad \Gamma \vdash_\Sigma \sigma =_{\beta_\mathcal{P}} \tau}{\Gamma \vdash_\Sigma M : \tau}\,(\mathsf{O{\cdot}Conv})$$

GLF's Equality

$\Gamma \vdash_\Sigma K =_{\beta_\mathcal{P}} K'$ K and K' are equal kinds in Γ and Σ

$\Gamma \vdash_\Sigma \sigma =_{\beta_\mathcal{P}} \tau$ σ and τ are equal types in Γ and Σ

$\Gamma \vdash_\Sigma M =_{\beta_\mathcal{P}} N$ M and N are equal terms in Γ and Σ

Type Equality Rule

$$\frac{\forall y_i \in \mathsf{Dom}(\widehat{\mathcal{P}}(M)).\ [\ \Gamma, \Delta \vdash_\Sigma \widehat{\mathcal{P}}(M)(y_i) : \Delta(y_i)\ ] \qquad \Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \qquad \Gamma \vdash_\Sigma M : \sigma}{\Gamma \vdash_\Sigma (\lambda \mathcal{P}{:}\Delta.\tau)\, M =_{\beta_\mathcal{P}} \tau\, \widehat{\mathcal{P}}(M)}$$

A Galleria of Desired Properties to get the Meta-Paradise ...

Subderivation Property

- Any derivation of $\Gamma \vdash_\Sigma \alpha$ has subderivations of Σ sig and $\vdash_\Sigma \Gamma$;
- Any derivation of $\Sigma, a{:}K$ sig has a subderivation of $\vdash_\Sigma K$;
- Any derivation of $\Sigma, f{:}\sigma$ sig has a subderivation of $\vdash_\Sigma \sigma$: Type;
- Any derivation of $\vdash_\Sigma \Gamma, x{:}\sigma$ has a subderivation of $\Gamma \vdash_\Sigma \sigma$: Type;
- Given a derivation of $\Gamma \vdash_\Sigma \alpha$ and any subterm occurring in the subject of the judgment, there exists a derivation of a smaller length of a judgment having that subterm as a subject;
- If $\Gamma \vdash_\Sigma \sigma : K$, then $\Gamma \vdash_\Sigma K$;
- If $\Gamma \vdash_\Sigma M : \sigma$, then $\Gamma \vdash_\Sigma \sigma$: Type.

Derivability of Weakening and Permutation

If Γ and Δ are valid contexts, and every declaration occurring in Γ also occurs in Δ, then $\Gamma \vdash_\Sigma \alpha$ implies $\Delta \vdash_\Sigma \alpha$.

Unicity of Types and Kinds

- If $\Gamma \vdash_\Sigma M : \sigma$ and $\Gamma \vdash_\Sigma M : \tau$, then $\Gamma \vdash \sigma =_{\beta_\mathcal{P}} \tau$;
- If $\Gamma \vdash_\Sigma \sigma : K$ and $\Gamma \vdash_\Sigma \sigma : K'$, then $\Gamma \vdash_\Sigma K =_{\beta_\mathcal{P}} K'$.

Transitivity or Cut

If $\Gamma, x{:}\sigma, \Delta \vdash_\Sigma \alpha$ and $\Gamma \vdash_\Sigma M : \sigma$, then $\Gamma, \Delta[M/x] \vdash_\Sigma \alpha[M/x]$.

Confluence

- If $K_1 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} K_2$ and $K_1 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} K_3$, then there exists K_4 such that $K_2 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} K_4$ and $K_3 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} K_4$;
- If $\sigma_1 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} \sigma_2$ and $\sigma_1 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} \sigma_3$, then there exists σ_4 such that $\sigma_2 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} \sigma_4$ and $\sigma_3 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} \sigma_4$;
- If $M_1 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} M_2$ and $M_1 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} M_3$, then there exists M_4 such that $M_2 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} M_4$ and $M_3 \mapsto\!\!\!\!\rightarrow_{\beta_\mathcal{P}} M_4$.

Abstraction Typing

- If $\Gamma \vdash_\Sigma \lambda\mathcal{P}{:}\Delta.\tau : \Pi\mathcal{P}{:}\Delta'.K$, then $\mathsf{Dom}(\Delta) \equiv \mathsf{Dom}(\Delta')$, and forall $x \in \mathsf{Dom}(\Delta)$, we have $\Gamma, \Delta \vdash_\Sigma \Delta(x) =_{\beta_\mathcal{P}} \Delta'(x)$;
- If $\Gamma \vdash_\Sigma \lambda\mathcal{P}{:}\Delta.M : \Pi\mathcal{P}{:}\Delta'.\tau$, then $\mathsf{Dom}(\Delta) \equiv \mathsf{Dom}(\Delta')$, and forall $x \in \mathsf{Dom}(\Delta)$, we have $\Gamma, \Delta \vdash_\Sigma \Delta(x) =_{\beta_\mathcal{P}} \Delta'(x)$;
- If $\Gamma \vdash_\Sigma \lambda\mathcal{P}{:}\Delta.\tau : \Pi\mathcal{P}{:}\Delta.K$, then $\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma$ and $\Gamma, \Delta \vdash_\Sigma \tau : K$;
- If $\Gamma \vdash_\Sigma \lambda\mathcal{P}{:}\Delta.M : \Pi\mathcal{P}{:}\Delta.\tau$, then $\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma$ and $\Gamma, \Delta \vdash_\Sigma M : \tau$.

Subject Reduction

- If $\Gamma \vdash_\Sigma K$ and $K \rightarrow_{\beta_\mathcal{P}} K'$, then $\Gamma \vdash_\Sigma K'$;
- If $\Gamma \vdash_\Sigma \sigma : K$ and $\sigma \rightarrow_{\beta_\mathcal{P}} \tau$, then $\Gamma \vdash_\Sigma \tau : K$;
- If $\Gamma \vdash_\Sigma M : \sigma$ and $M \rightarrow_{\beta_\mathcal{P}} N$, then $\Gamma \vdash_\Sigma N : \sigma$.

Strong Normalization

- If $\Gamma \vdash_\Sigma K$, then K is strongly normalizing;
- If $\Gamma \vdash_\Sigma \sigma : K$, then σ is strongly normalizing;
- If $\Gamma \vdash_\Sigma M : \sigma$, then M is strongly normalizing.

Judgments decidability

It is decidable whether $\Gamma \vdash_\Sigma \alpha$ is derivable.

Back to the Earth ...

The following is about the **most** that one could prove for a General Logical Framework at this stage of generality.

1. **The Subderivation Property is valid in GLF;**

2. **Derivability of Weakening and Permutation is valid in GLF;**

3. **Unicity of Types and Kinds is valid in GLF;**

4. **If Abstraction Typing is valid, then Subject Reduction is valid in GLF.**

First Example - The Closed Logical Framework CLF

CLF is an instance of GLF obtained by considering the following two unary predicates and corresponding **Hat** and **Bar** functions:

$$\mathsf{True}(M) \triangleq true \;\; (\forall M)$$

$$\mathsf{Closed}(M) \triangleq \begin{cases} true & \text{if } \mathsf{Fv}(M) = \varnothing \\ false & \text{otherwise} \end{cases}$$

$$\overline{\mathsf{True}, \langle x{:}A\rangle} = \overline{\mathsf{Closed}, \langle x{:}A\rangle} = x$$

$$\widehat{\mathsf{True}, \langle x{:}A\rangle}, N = \widehat{\mathsf{Closed}, \langle x{:}A\rangle}, N = [N/x]$$

One can easily see that $\lambda\mathsf{True}{:}\langle x{:}A\rangle.M$ corresponds to the standard LF term $\lambda x{:}A.M$. We will write $\lambda\mathsf{Closed}{:}\langle x{:}A\rangle.M$ simply as $\lambda_{\varnothing} x{:}A.M$.

Hence CLF features the following two notions of β-reduction:

1. **standard** β-reduction: $(\lambda x{:}A.M)\, N \rightarrow_\beta M[N/x]$
2. **closed** β-reduction: $(\lambda_{\varnothing} x{:}A.M)\, N \rightarrow_\beta M[N/x]$ if $\mathsf{Fv}(N) = \varnothing$

The Properties of CLF

Standard β-reduction and closed β-reduction nicely combine, in the sense that a potential closed β-reduction is preserved under application of any substitution (coming from another, possibly standard reduction), and vice versa:

Lemma 1 (Closure under Reduction and Substitution) *If $M \in \Lambda^{\varnothing}$, then, for any substitution θ, $N\theta \in \Lambda^{\varnothing}$. Moreover, for any M and N, and for any θ such that $x \notin \mathsf{CoDom}(\theta)$, we have $N[M/x]\theta \equiv N\theta[M\theta/x]$.*

This is the basis for proving

Theorem 1 (Confluence) *The relation $\mapsto_\beta$ is confluent.* □

Proof. Using a suitable notion of parallel reduction.

Strong Normalization follows from strong normalization of LF

Subject Reduction, and **Typing Decidability** also hold

Application: Σ_{S_4} for classic S_4 modal logic in Hilbert style in CLF

Propositional Connectives and Judgment

o : Type $\supset$: o^3 $\neg$: o^2 $\Box$: o^2 True : $o \rightarrow$ Type

Propositional Axioms

A_1 : $\Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}\phi \supset (\psi \supset \phi)$

A_2 : $\Pi\phi^o.\ \Pi\psi^o.\ \Pi\theta^o.\ \mathsf{True}(\phi \supset (\psi \supset \theta)) \supset (\phi \supset \psi) \supset (\phi \supset \theta)$

A_3 : $\Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}(\neg\psi \supset \neg\phi) \supset ((\neg\psi \supset \phi) \supset \psi$

Modal Axioms

K : $\Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}\Box(\phi \supset \psi) \supset (\neg\phi \supset \neg\psi)$

4 : $\Pi\phi^o.\ \mathsf{True}\Box\phi \supset \Box\Box\phi$

$\top$: $\Pi\phi^o.\ \mathsf{True}\Box\phi \supset \phi$

Rules

MP : $\Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}\phi \supset \mathsf{True}(\phi \supset \psi) \rightarrow \mathsf{True}\psi$

NEC : $\Pi\phi^o.\ \Pi_{\varnothing} x{:}\mathsf{True}\phi.\ \mathsf{True}\Box\phi$

Second Example - The Pattern Logical Framework

Terms

$$\Sigma ::= \varnothing \mid \Sigma, a{:}K \mid \Sigma, f{:}A \qquad \text{Signatures}$$

$$\Gamma, \Delta ::= \varnothing \mid \Gamma, x{:}A \qquad \text{Contexts}$$

$$K ::= \mathsf{Type} \mid \Pi P{:}\Delta.K \mid \lambda P{:}\Delta.K \mid K\ M \qquad \text{Kinds}$$

$$A, B, C ::= a \mid \Pi P{:}\Delta.A \mid \lambda P{:}\Delta.A \mid A\ M \qquad \text{Families}$$

$$M, N, Q ::= f \mid x \mid \lambda P{:}\Delta.M \mid M\ M \qquad \text{Objects}$$

where $P \in \mathcal{O}_{\mathcal{P}} \subseteq \mathcal{O}$ and $\mathcal{O}_{\mathcal{P}}$ is a set of **patterns** to be defined.

Safe terms: the free variables occurring in patterns are precisely the variables declared in the corresponding context.

β-reduction

The pattern-matching algorithm can either **fire a substitution**, or keep the **computation stuck**, unless further substitutions are provided.

E.g., for an algebraic constant f of type $a \to a$,

$$M \equiv (\lambda(f\, y){:}[y{:}a].y)\, x$$

is stuck, but

$$(\lambda(f\, x){:}[x{:}a].M)\, (f\, (f\, 3)) \mapsto\!\!\!\!\to_\beta 3$$

In general,

$$(\lambda P{:}\Delta.M)\, N \mapsto_\beta M\theta \quad \text{if } \theta = \mathcal{A}lg(P; N)$$

where $\mathcal{A}lg$ (omitted in this talk, simplification of POPL03) is such that,

$$\text{if } \theta = \mathcal{A}lg(P; N), \text{ then } P\theta \equiv N$$

PLF - Critical design issues: Shape of Patterns

Let $\mathcal{O}_{\mathcal{P}}$ be the set of patterns defined by

$$\mathcal{O}_{\mathcal{P}} \triangleq \left\{ P \in \mathsf{Nf}_{\mathcal{O}} \mid \begin{array}{l} \mathsf{LPC}(P; \mathsf{Fv}(P)) = \mathsf{true} \ \wedge \\ \mathsf{APC}(P; \mathsf{Fv}(P)) = \mathsf{false} \end{array} \right\}$$

LPC expresses the **linear pattern** condition, APC expresses the **active pattern** condition.
Why so complicated?

PLF - Critical design issue 1

- **Variables in functional position.** It is well known since van Ostroom '90, that allowing variables in functional position breaks confluence.

$$M \triangleq (\lambda(x\,y){:}[x{:}a{\to}a, y{:}a].x)\ (\mathsf{I}\ z)$$

where

$$\mathsf{I} \triangleq \lambda x{:}a.x$$

Namely,

$$M \mapsto_\beta (\lambda(x\,y){:}[x{:}a{\to}a, y{:}a].x)\ z$$

by reducing the argument, while

$$M \mapsto_\beta \mathsf{I}$$

by reducing the outermost redex.

PLF - Critical design issue 2

- **Linearity condition.** It is also well-known since Klop '80, that if we abandon the linearity condition in patterns, we loose confluence of raw terms. Namely, let
  - $Y \triangleq (\lambda y{:}?.\lambda x{:}?.(x\,(y\,y\,x)))\,(\lambda y{:}?.\lambda x{:}?.(x\,(y\,y\,x)))$ be the (hopefully untypable) fix-point combinator
  - $D \triangleq \lambda(f\ z\ z){:}[z{:}a].e$ be a term with a non-linear pattern
  - $C \triangleq Y\,(\lambda y{:}?.\lambda x{:}?.D\,(f\,x\,(y\,x)))$
  - $A \triangleq Y\,C$

  Then, we have: $A \mapsto\!\!\!\!\rightarrow_\beta C\,e$ and $A \mapsto\!\!\!\!\rightarrow_\beta e$. Thus the system is not confluent. However, one can check that the fix-point operator Y is *not* typable. Hence the above counterexample does not apply to the case of well-typed (strongly normalizing) terms

PLF - Critical design issue 3

- **Substitution-stuck redexes.** The reason for allowing in patterns only substitution-stuck redexes, and not simply stuck redexes, is that, in this way, patterns can match only arguments where the corresponding redexes will never fire. Otherwise, if we include patterns of the shape $(\lambda P_1{:}\Delta.P_2)\ P_3\ \vec{P}'$, where only $\mathcal{A}lg(P_1; P_3) = \mathsf{fail}$, *i.e.* only the present reduction is stuck, we loose confluence. A counterexample

$$M \triangleq (\lambda((\lambda \mathsf{I}{:}\varnothing.\mathsf{I})\ x){:}[x{:}a{\to}a].x)\ ((\lambda \mathsf{I}{:}\varnothing.\mathsf{I})\ \mathsf{I})$$

Reducing the outermost redex, we get $M \mapsto_\beta \mathsf{I}$; reducing inside the argument,

$$M \mapsto_\beta (\lambda((\lambda \mathsf{I}{:}\varnothing.\mathsf{I})\ x){:}[x{:}a{\to}a].x)\ \mathsf{I}$$

PLF - Critical design issue 4

- **Exact Pattern Condition.** We consider only terms where the variables occurring in patterns are precisely the variables declared in the corresponding contexts.

$$\mathsf{Fv}(P) = \mathsf{Dom}(\Delta)$$

On the other hand, one could think of having $\mathsf{Dom}(\Delta) \subseteq \mathsf{Fv}(P)$, *i.e.* patterns can contain free variables, which can be bound outside, and hence they can be substituted during reductions (a.k.a. **dynamic patterns**), as the variable y in the following term

$$(\lambda y{:}a.\lambda(f\ x\ y){:}[x{:}a].y)\ z \mapsto_\beta \lambda(f\ x\ z){:}[x{:}a].z$$

But this causes problems when combined with untypable fix-points (see the encoding of the Klop counter example in the nice Wack Ph.D. 06), since non-linear terms can be mimicked via dynamic patterns, even under the linearity pattern condition

PLF - Critical design issue 5

- **Pattern reductions.** The counterexample in issue 3 above also shows that extending the class of patterns beyond normal forms, by allowing reductions in patterns is potentially dangerous.

  In this perspective, in order to preserve confluence when reductions in patterns are permitted, a possible solution is that of allowing reductions to fire only when the pattern is a normal form.

  This corresponds to partially fixing a reduction strategy.

  However, *K-reductions* in patterns deserve special discussion.

PLF - Critical design issue 6

- **K-reductions in patterns.** A *K-redex* is a redex $(\lambda P{:}\Delta.M)\ N$, where $\lambda P{:}\Delta.M$ is a *K-abstraction*, *i.e.*

$$\mathsf{Fv}(M) \subset \mathsf{Fv}(P)$$

When a K-redex is reduced, (parts of) the argument is erased. As a consequence, the Exact Pattern Condition is violated, and bound variables may become free. Example:

$$M \triangleq (\lambda \underbrace{((\lambda x{:}a.y)\ z)}_{P}{:}[y{:}a{\to}a, z{:}a].y\ z)\ \underbrace{((\lambda x{:}a.f)\ g)}_{N}$$

Then, by reducing the pattern P and the argument N, and then reducing the outermost redex, we get

$$M \mapsto\!\!\!\!\to_\beta (\lambda y{:}[y{:}a{\to}a, z{:}a].y\ z)\ f \mapsto_\beta f\ z$$

i.e. z comes out of its scope!

PLF - Still Critical design issue 6

To avoid this problem, we could simply **block K-reductions in patterns**, but then we also need to **block pattern matching when the pattern contains a K-redex**. Otherwise, we loose confluence, the term $M \stackrel{\triangle}{=} (\lambda \underbrace{((\lambda x{:}a.y)\ z)}_{P} {:}[y{:}a{\to}a, z{:}a].y\ z)\ \underbrace{((\lambda x{:}a.f)\ g)}_{N}$ above being a counterexample.

Namely, by reducing the outermost redex,

$$M \mapsto_\beta f\ g$$

while, by reducing the argument N, we get

$$M \mapsto_\beta (\lambda((\lambda x{:}a.y)\ z){:}[y{:}a{\to}a, z{:}a].y\ z)\ f$$

which is not reducible anymore

PLF - Results

Confluence

The relation $\mapsto_\beta$ is confluent.

Subject Reduction

1. If $\Gamma \vdash_\Sigma A{:}K$ and $A \mapsto_\beta B$, then $\Gamma \vdash_\Sigma B{:}K$;
2. If $\Gamma \vdash_\Sigma M : A$ and $M \mapsto_\beta N$, then $\Gamma \vdash_\Sigma N : A$.

Strong Normalization

1. If $\Gamma \vdash_\Sigma K$, then $K \in \mathsf{SN}^{\mathcal{K}}$;
2. If $\Gamma \vdash_\Sigma A : K$, then $A \in \mathsf{SN}^{\mathcal{F}}$;
3. If $\Gamma \vdash_\Sigma M : A$, then $M \in \mathsf{SN}^{\mathcal{O}}$.

Judgments decidability

It is decidable whether the PLF judgment $\Gamma \vdash_\Sigma \alpha$ is derivable.

PLF is particularly suitable as a Logical Framework for encoding **case analysis**

Applications: Σ_v for Plotkin's λ_v-calculus in PLF

Syntactic Categories

o : Type

Constructors and Judgments

$! : o^2$ Lam : $\Pi f{:}[\Pi !x^o.o].\ o$ App : o^3 $= : o \to o \to$ Type

Axioms and Rules

$\mathsf{Eq}_{\mathsf{refl}}$: $\Pi x^o.\ x = x$

$\mathsf{Eq}_{\mathsf{symm}}$: $\Pi x^o.\ \Pi y^o.\ (x = y) \to (y = x)$

$\mathsf{Eq}_{\mathsf{trans}}$: $\Pi x^o.\ \Pi y^o.\ \Pi z^o.\ (x = y) \to (y = z) \to (x = z)$

$\mathsf{Eq}_{\mathsf{ctx}}$: $\Pi x^o.\ \Pi y^o.\ \Pi z^o.\ \Pi w^o.\ (x = y) \to (z = w) \to (\mathsf{App}\ x\ z = \mathsf{App}\ y\ w)$

Betav : $\Pi f{:}[\Pi !x^o.o].\ \Pi y^o.\ \mathsf{App}\ (!\,(\mathsf{Lam}\ f))\ (!\,y) = f\ (!\,y)$

Xiv : $\Pi f{:}[\Pi !x^o.o].\ \Pi g{:}[\Pi !x^o.o].$

$(\Pi z^o.\ f\ (!\,z) = g\ (!\,z) \to (!\,(\mathsf{Lam}\ f) = !\,(\mathsf{Lam}\ g))$

Etav : $\Pi x^o.\ !\,(\mathsf{Lam}\ (\lambda(!\,y^o).\mathsf{App}\ (!\,x)\ (!\,y))) = !\,x$

Questions and future work

- Is there a **semantics** for $\lambda\mathcal{P}{:}\Delta.M$? Using monads?
- Are there interesting **Curry-Howard isomorphisms** for GLF or more specifically for systems blending rewriting facilities and higher order calculi, such as PLF?
- Extend existing proof assistants based on dependent type systems, *e.g*. Coq, with pattern matching facilities as in PLF, and more generally with GLF.
- Instantiate GLF in order to give sharp encodings of relevance and linear logics?
- We have seen that here is no strong notion of pattern reduction. Still, can we allow reductions in patterns under specific strategies?
- Can the linearity restriction in patterns be relaxed, still preserving confluence and strong normalization over well typed patterns?
- Our results should scale up to all the systems of **rho-cube** *à la* Cirstea-Kirchner-Liquori (FOSSACS-01,POPL-03), *i.e*. to systems corresponding to the full Calculus of Constructions and Pure Pattern Type Systems.
- Formalize the notion of predicate $\mathcal{P}$, still preserving generality.

- **Confluence** and **Strong Normalization** should hold for a generic predicate calculus, provided that the various notions of reductions nicely combine, in the sense that $\mapsto_{\mathcal{P}_i}$-reductions are preserved both under $\mapsto_{\mathcal{P}_j}$-reductions of the argument and application to the argument of any substitution coming from other reductions.

- **Case analysis** in PLF should be compared with that of inductive types in Coq.

- Instantiate GLF so as to provide a more natural encoding of the Natural Deduction ***□-introduction rule*** of Prawitz:

$$\frac{\Box\Gamma \vdash \phi}{\Box\Gamma \vdash \Box\phi}(\Box\text{-}Intro)$$

E.g. if we introduce a new predicate $\mathsf{Occurs}_x \stackrel{\triangle}{=}$ "x is a term whose free variables occur only in subterms of type $\mathsf{True}\Box\psi$ for some ψ", then $\Box\text{-}Intro$ becomes:

$$\Box\text{-}I : \Pi\phi{:}o.\ \Pi\mathsf{Occurs}_x{:}[x{:}\mathsf{True}\phi].\ \mathsf{True}\Box\phi.$$

Rewriting Calculus Statistics

- **New** syntax and **New** Operational Semantics in 2001 (RTA) almost constant since 2006 (modulo symbol choices and one redundant-innocuous one-step rule (σ)
- Type system**s** for Rho set up in 2001 (FOSSACS) and **almost constant** since 2006 (modulo some symbol choices)
- **Syntax Restrictions on Patterns vary since 1998!**
- **Dynamic Patterns** in 2003 (POPL)
- **Strong Normalization** conjectured in 2001, achieved in 2004 for the simply types with "minipatterns" (*à la* Wack) via compilation to F_ω, achieved in 2006 using Tait & Löf
- 20 trip to Nancy, Sophia, Udine, Edinburgh, 1 PhD, 5 diff. techniques, 2 rejection, circa 300 CVS commit, 5-7 people involved, circa 1000 expressos, etc.
- In one sentence: ***The most expensive proof of my 15YY academic life!!***

How to get a copy of this paper

- In Gordon Plotkin's Festschrift, ENTCS

- Now on `hal.inria.fr`, the INRIA's open archive

Comp Sets

- Let $\mathsf{Comp}^{\mathcal{O}}$ be the set of *object computability candidates* defined as follows.

  $\mathcal{N} \in \mathsf{Comp}^{\mathcal{O}}$ if and only if $\mathcal{N}$ satisfies:

  (c1) $\mathcal{N} \subseteq \mathsf{SN}^{\mathcal{O}}$;

  (c2) $\forall \vec{N} \in \mathsf{SN}^{\mathcal{O}}.\ x\,\vec{N}$, and $f\,\vec{N} \in \mathcal{N}$;

  (c3) $\mathcal{N}$ is closed under the rule

$$\frac{Q \mapsto\!\!\!\!\rightarrow_{\beta} Q' \quad \mathcal{A}lg(P; Q') = \theta \quad (M\theta)\vec{N} \in \mathcal{N} \quad \mathsf{CoDom}(\Delta), Q \in \mathsf{SN}}{(\lambda P{:}\Delta.M)\,Q\,\vec{N} \in \mathcal{N}}$$

  (c4) $\mathcal{N}$ is closed under the rule

$$\frac{\forall Q'.\ [Q \mapsto\!\!\!\!\rightarrow_{\beta} Q' \Rightarrow \mathcal{A}lg(P; Q') = \mathsf{fail}] \quad \mathsf{CoDom}(\Delta), M, Q, \vec{N} \in \mathsf{SN}}{(\lambda P{:}\Delta.M)\,Q\,\vec{N} \in \mathcal{N}}$$

- Let $\mathsf{Comp}^{\mathcal{F}}$ be the set of *family computability candidates* defined as follows.

$\mathcal{N} \in \mathsf{Comp}^{\mathcal{F}}$ if and only if $\mathcal{N}$ satisfies:

(c1) $\mathcal{N} \subseteq \mathsf{SN}^{\mathcal{F}}$;

(c2) $\forall \vec{N} \in \mathsf{SN}^{\mathcal{F}}.\ a\,\vec{N} \in \mathcal{N}$;

(c3) $\mathcal{N}$ is closed under the rule

$$\frac{Q \mapsto\!\!\!\!\rightarrow_{\beta} Q' \quad \mathcal{A}lg(P; Q') = \theta \quad (A\theta)\vec{N} \in \mathcal{N} \quad \mathsf{CoDom}(\Delta), Q \in \mathsf{SN}}{(\lambda P{:}\Delta.A)\,Q\,\vec{N} \in \mathcal{N}}$$

(c4) $\mathcal{N}$ is closed under the rule

$$\frac{\forall Q'.\ [Q \mapsto\!\!\!\!\rightarrow_{\beta} Q' \Rightarrow \mathcal{A}lg(P; Q') = \mathsf{fail}] \quad \mathsf{CoDom}(\Delta), A, Q, \vec{N} \in \mathsf{SN}}{(\lambda P{:}\Delta.A)\,Q\,\vec{N} \in \mathcal{N}}$$

Type and Kind Interpretation

- Let $[\![-]\!]^{\mathcal{F}}$ be the family interpretation function defined by induction on families as follows.

$$[\![a\,\vec{N}]\!]^{\mathcal{F}} = \mathsf{SN}^{\mathcal{O}}$$

$$[\![\checkmark P{:}\Delta{:}B.A]\!]^{\mathcal{F}} =$$

$$\left\{ M \,\middle|\, Q \in [\![B]\!]^{\mathcal{F}} \implies M\,Q \in \begin{cases} \mathsf{SN}^{\mathcal{O}} & \text{if } P \not\sqsubseteq Q \\ \bigcup\{ [\![A\theta]\!]^{\mathcal{F}} \mid Q \twoheadrightarrow_{\beta} Q' \wedge \theta = \mathcal{A}lg(P; Q') \} & \text{otherwise} \end{cases} \right\}$$

$$[\![(\lambda P{:}\Delta.A)\, M\, \vec{N}]\!]^{\mathcal{F}} = \begin{cases} \mathsf{SN}^{\mathcal{O}} & \text{if } P \not\sqsubseteq M \\ \bigcup\{ [\![(A\theta)\vec{N}]\!]^{\mathcal{F}} \mid M \twoheadrightarrow_{\beta} M' \wedge \theta = \mathcal{A}lg(P; M') \} & \text{otherwise} \end{cases}$$

- Let $[\![-]\!]^{\mathcal{K}}$ be the family interpretation function defined by induction on kinds as follows.

$$[\![\mathsf{Type}\,\vec{N}]\!]^{\mathcal{K}} = \mathsf{SN}^{\mathcal{F}}$$

$$[\![\checkmark P{:}\Delta{:}B.K]\!]^{\mathcal{K}} =$$

$$\left\{ A \,\middle|\, Q \in [\![B]\!]^{\mathcal{K}} \implies A\,Q \in \begin{cases} \mathsf{SN}^{\mathcal{F}} & \text{if } P \not\sqsubseteq Q \\ \bigcup\{ [\![K\theta]\!]^{\mathcal{K}} \mid Q \twoheadrightarrow_{\beta} Q' \wedge \theta = \mathcal{A}lg(P; Q') \} & \text{otherwise} \end{cases} \right\}$$

$$[\![(\lambda P{:}\Delta.K)\, M\, \vec{N}]\!]^{\mathcal{K}} = \begin{cases} \mathsf{SN}^{\mathcal{F}} & \text{if } P \not\sqsubseteq M \\ \bigcup\{ [\![(K\theta)\, \vec{N}]\!]^{\mathcal{K}} \mid M \twoheadrightarrow_{\beta} M' \wedge \theta = \mathcal{A}lg(P; M') \} & \text{otherwise} \end{cases}$$

Klop's Counter Example in Barry's Pattern Calculus

$$\Omega \triangleq [y]y \to [x]x \to (x\,(y\,y\,x))$$

$$Y \triangleq \Omega\,\Omega$$

$$D \triangleq [x,y](d\,x\,y) \to ([\,]x \to e) \qquad \text{Morally } [x](d\,x\,x) \to e$$

$$E \triangleq ([y]y \to [x]x \to D\,(d\,x\,(y\,x)))$$

$$C \triangleq Y\,E \qquad \text{Reduces to } \Rightarrow E\,C \Rightarrow [x]x \to D\,(d\,x\,(C\,x))$$

$$A \triangleq Y\,C$$

$$A \Rightarrow C\,A \Rightarrow D\,(d\,A\,(C\,A)) \Rightarrow D\,(d\,(C\,A)\,(C\,A))$$

$$\Downarrow \qquad\qquad\qquad\qquad\qquad\qquad \Downarrow$$

$$C\,e \qquad\qquad\qquad\qquad\qquad\qquad e$$