# A Coq-definitional implementation of the Lax Logical Framework $\mathsf{LLF}_{\mathcal{P}}$, for "fast and loose" reasoning

**F. Alessi, A. Ciaffaglione, P. Di Gianantonio, F. Honsell, M. Lenisa**
name.surname@uniud.it
**Department of Mathematics, Computer Science, and Physics**
**University of Udine - Udine, Italy**

**Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP-2019)**
Vancouver, Canada - June 22, 2019

We are grateful to **Ivan Scagnetto** and anonymous referees for helpful comments and suggestions

# Outline

1. Motivation
2. The Logical Frameworks $\text{LLF}_{\mathcal{P}}$ and $\text{LLF}_{\mathcal{P}}{}^{+}$
3. The monadic nature of Locks in $\text{LLF}_{\mathcal{P}}$ and $\text{LLF}_{\mathcal{P}}{}^{+}$
4. Applications of Locks
5. Implementation of $\text{LLF}_{\mathcal{P}}$ and $\text{LLF}_{\mathcal{P}}{}^{+}$ in Coq
6. Call-by-value $\lambda$-calculus
7. Branch prediction
8. Optimistic concurrency control

# Motivation for $\mathrm{LLF}_{\mathcal{P}}$'s

Prudentially and incrementally, **extend conservatively** LF so as to:

- **integrate** in a unique Logical Framework, *different epistemic sources* of **evidence** deriving from special-purpose tools, oracles, and even *non-apodictic* ones *e.g.* explicit computations, deduction up-to, diagrams, physical analogies;
- **factor-out, postpone, run in parallel** the verification of "morally" *proof-irrelevant* and *time-consuming* judgments and **side conditions**;
- for **supporting** formal reasoning according to the **fast and loose reasoning paradigm**, which trades off *correctness* for *efficiency*, by running in parallel computationally demanding checks, or postponing tedious verifications until worthwhile.
- This paradigm is used
  - in **everyday mathematics** carried out in *naïve* Set Theory, or when introducing blanket assumptions to be formalized and checked later, *e.g.* *typical ambiguity* $\mathcal{U} \in \mathcal{U}$;
  - in **branch prediction** in *processor architecture* or **optimistic concurrency** in *distributed systems*.
- $\mathrm{LLF}_{\mathcal{P}}$'s appear in a series of papers by subsets of the authors and also I.Scagnetto, L.Liquori, and P.Maksimović since 2007 [8,9,10,11]. $\mathrm{LLF}_{\mathcal{P}}$'s were presented at LFMTP in 2012-13-15-17.

# The LLF$_{\mathcal{P}}$ Logical Framework

- Syntax

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

- Reduction

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

- Typing judgments

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

# The extended $\mathrm{LLF}_{\mathcal{P}}{}^{+}$ Logical Framework

- Syntax

| | | | | |
|---|---|---|---|---|
| $\Sigma \in \mathcal{S}$ | $\Sigma$ | $::=$ | $\emptyset \mid \Sigma, a{:}K \mid \Sigma, c{:}\sigma$ | *Signatures* |
| $\Gamma \in \mathcal{C}$ | $\Gamma$ | $::=$ | $\emptyset \mid \Gamma, x{:}\sigma$ | *Contexts* |
| $K \in \mathcal{K}$ | $K$ | $::=$ | $\mathsf{Type} \mid \Pi x{:}\sigma.K$ | *Kinds* |
| $\sigma, \tau, \rho \in \mathcal{F}$ | $\sigma$ | $::=$ | $a \mid \Pi x{:}\sigma.\tau \mid \sigma\, N \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \mid \mathcal{L}^{\mathcal{P}}_{\sigma,\mathbf{K}}[\rho]$ | *Families* |
| $M, N \in \mathcal{O}$ | $M$ | $::=$ | $c \mid x \mid \lambda x{:}\sigma.M \mid M\, N \mid$ | |
| | | | $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \mid \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \mid \mathcal{L}^{\mathcal{P}}_{\sigma,K}[M] \mid \mathcal{U}^{\mathcal{P}}_{\sigma,K}[M]$ | *Objects* |

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

$$\mathcal{U}^{\mathcal{P}}_{U,V}[\mathcal{L}^{\mathcal{P}}_{U,V}[W]] \to_{\beta\mathcal{L}} W \qquad \mathcal{L}^{\mathcal{P}}_{U,V}[\mathcal{U}^{\mathcal{P}}_{U,V}[W]] \to_{\beta\mathcal{L}} W$$

- Typing judgments

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

# LLF$_{\mathcal{P}}$'s typing rules (objects)

The crucial rules are those dealing with lock types:

- lock-introduction

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

- lock-elimination

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

- guarded lock-elimination

$$\frac{\begin{array}{l}\Gamma, x{:}\tau \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{S,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho] \\ \Gamma \vdash_\Sigma N : \mathcal{L}^{\mathcal{P}}_{S',\sigma'}[\tau] \quad \sigma =_{\beta\mathcal{L}} \sigma' \quad S =_{\beta\mathcal{L}} S'\end{array}}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{S,\sigma}[M[\mathcal{U}^{\mathcal{P}}_{S',\sigma'}[N]/x]] : \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho[\mathcal{U}^{\mathcal{P}}_{S',\sigma'}[N]/x]]} \; (O{\cdot}Guarded{\cdot}Unlock)$$

# Extended LLF$_{\mathcal{P}}{}^{+}$'s typing rules

Locks can access all sorts of judgments $\Gamma \vdash U : V$:

- lock-introduction

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

- un-guarded lock-elimination

$$\frac{\begin{array}{c}\Gamma, x{:}\tau \vdash_\Sigma M : \rho \qquad \Gamma \vdash_\Sigma N : \mathcal{L}^{\mathcal{P}}_{U,V}[\tau] \\ \mathcal{P}(\Gamma \vdash_\Sigma U' : V') \quad V {=_{\beta\mathcal{L}}} V' \quad U {=_{\beta\mathcal{L}}} U'\end{array}}{\Gamma \vdash_\Sigma M[\mathcal{U}^{\mathcal{P}}_{U',V'}[N]/x] : \rho[\mathcal{U}^{\mathcal{P}}_{U',V'}[N]/x]} \ (O{\cdot}Top{\cdot}Unlock)$$

- guarded lock-elimination

$$\frac{\begin{array}{c}\Gamma, x{:}\tau \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{U',V'}[\rho] \qquad \Gamma \vdash_\Sigma N : \mathcal{L}^{\mathcal{P}}_{U,V}[\tau] \\ V {=_{\beta\mathcal{L}}} V' \quad U {=_{\beta\mathcal{L}}} U'\end{array}}{\Gamma \vdash_\Sigma M[\mathcal{U}^{\mathcal{P}}_{U',V'}[N]/x] : \mathcal{L}^{\mathcal{P}}_{U',V'}[\rho][\mathcal{U}^{\mathcal{P}}_{U',V'}[N]/x]} \ (O{\cdot}Guarded{\cdot}Unlock)$$

# $\mathrm{LLF}_{\mathcal{P}}{}^{+}$'s typing rules (signatures, contexts, kinds, families)

**Valid signatures**

$$\frac{}{\emptyset \text{ sig}} \ (S{\cdot}Empty)$$

$$\frac{\vdash_\Sigma K \quad a \notin \mathrm{Dom}(\Sigma)}{\Sigma, a{:}K \text{ sig}} \ (S{\cdot}Kind)$$

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

**Context rules**

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

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

**Kind rules**

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

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

**Family rules**

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

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

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

$$\frac{\Gamma \vdash_\Sigma \sigma : K \quad \Gamma \vdash_\Sigma K' \quad K {=}_{\beta\mathcal{L}} K'}{\Gamma \vdash_\Sigma \sigma : K'} \ (F{\cdot}Conv)$$

$$\frac{\Gamma \vdash_\Sigma \rho : \mathrm{Type} \quad \Gamma \vdash_\Sigma U : V}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{U,V}[\rho] : \mathrm{Type}} \ (F{\cdot}Lock)$$

$$\frac{\begin{array}{l}\Gamma, x{:}\tau \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{U,V}[\rho] : \mathrm{Type} \\ \Gamma \vdash_\Sigma N : \mathcal{L}^{\mathcal{P}}_{U',V'}[\tau] \\ U {=}_{\beta\mathcal{L}} U' \quad V {=}_{\beta\mathcal{L}} V'\end{array}}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{U,V}[\rho[\mathcal{U}^{\mathcal{P}}_{U',V'}[N]/x]] : \mathrm{Type}} \ (F{\cdot}Guarded{\cdot}Unlock)$$

# LLF$_\mathcal{P}$'s formal properties

- strong normalization
- confluence
- subject reduction (for well-behaved predicates)

**Definition (Well-behaved predicates)**

A finite set of predicates $\{\mathcal{P}_i\}_{i\in I}$ is well-behaved if each $\mathcal{P}$ 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 $\mathcal{P}(\Gamma \vdash_\Sigma \alpha)$ holds, then $\mathcal{P}(\Delta \vdash_\Omega \alpha)$ also holds.

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

Closure under reduction. If $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$ holds and $N \to_{\beta\mathcal{L}} N'$ $(\sigma \to_{\beta\mathcal{L}} \sigma')$ holds, then $\mathcal{P}(\Gamma \vdash_\Sigma N' : \sigma)$ $(\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma'))$ also holds.

# The monadic nature of and $\mathrm{LLF}_{\mathcal{P}}$ and $\mathrm{LLF}_{\mathcal{P}}{}^{+}$

- for each $U, V$ such that $\Gamma \vdash U : V$ and well behaved $\mathcal{P}$ the operator $\mathcal{L}^{\mathcal{P}}_{U,V}[\_]$ induces a **strong monad**, or equivalently a **Kleisli triple**, once we view the **Term Model** of $\mathrm{LLF}_{\mathcal{P}}$ as a category;
- the monad $(T_{\mathcal{P}}, \eta, \mu)$ is given by
  - $\eta \triangleq \lambda x : \rho.\ \mathcal{L}^{\mathcal{P}}_{U,V}[x] \ : \ \rho \to \mathcal{L}^{\mathcal{P}}_{U,V}[\rho]$
  - $\mu \triangleq \lambda x : \mathcal{L}^{\mathcal{P}}_{U,V}[\mathcal{L}^{\mathcal{P}}_{U,V}[\rho]].\ \mathcal{L}^{\mathcal{P}}_{U,V}[\mathcal{U}^{\mathcal{P}}_{U,V}[\mathcal{U}^{\mathcal{P}}_{U,V}[x]]] \ : \ \mathcal{L}^{\mathcal{P}}_{U,V}[\mathcal{L}^{\mathcal{P}}_{U,V}[\rho]] \to \mathcal{L}^{\mathcal{P}}_{U,V}[\rho]$;
- the **guarded-unlock** rules "morally" amount to **Kleisli-composition**, namely, we can define an operator $\mathbf{let}_{\mathcal{P},U,V} : (\sigma \to \mathcal{L}^{\mathcal{P}}_{U,V}[\tau]) \to \mathcal{L}^{\mathcal{P}}_{U,V}[\sigma] \to \mathcal{L}^{\mathcal{P}}_{U,V}[\tau]$ as

$$\lambda x : \sigma \to \mathcal{L}^{\mathcal{P}}_{U,V}[\tau].\ \lambda y : \mathcal{L}^{\mathcal{P}}_{U,V}[\sigma].\ x(\mathcal{U}^{\mathcal{P}}_{U,V}[y]) : (\sigma \to \mathcal{L}^{\mathcal{P}}_{U,V}[\tau]) \to \mathcal{L}^{\mathcal{P}}_{U,V}[\sigma] \to \mathcal{L}^{\mathcal{P}}_{U,V}[\tau];$$

- the $\mathtt{let}_{\mathcal{P},U,V}$ constructor could be taken as primitive instead of $\mathcal{U}^{\mathcal{P}}_{U,V}[\ ]$, but then it should be extended also to types in the $F{\cdot}Guarded{\cdot}Unlock$ rule;
- the **monad equalities** hold:
  - $\mathcal{L}^{\mathcal{P}}_{U,V}[\ ]$ induces a *congruence*,
  - $\mathrm{LLF}_{\mathcal{P}}{}^{+}$ *reduction rules* amount to $T.\beta$ an $T.\eta$;
  - *associativity of Kleisli composition* holds by computation, namely for terms $Q, N, P$ of appropriate types both $\mathbf{let}_{\mathcal{P}} Q(\mathbf{let}_{\mathcal{P}} NM)$ and $\mathbf{let}_{\mathcal{P}}(\mathbf{let}_{\mathcal{P}} QN)M$ reduce to $\lambda x : \tau.Q(\mathcal{U}^{\mathcal{P}}_{U,V}[N(\mathcal{U}^{\mathcal{P}}_{U,V}[Mx])])$.

# Applications of Locks: side-conditions are *MONADS*

- **modal** logics: a proof term is *closed*;
- **substructural** logics *e.g. affine elementary linear* logic, *non-commutative linear logic*: variables in proof terms are constrained appropriately;
- **Hoare's** logic: *quantifier-free* formulæ, and *non-interference* predicates;
- **Fitch-Prawitz** Set Theory: proof terms are *normalizable*;
- **Poincaré**'s principle: terms are computationally(*definitionally*) equivalent;
- **Deduction Modulo**,

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

- **reasoning on totality**;
- reasoning and programming **up-to equivalence relations.**

# Applications of Locks: Squash types (after HOTT)

the operator $\mathcal{L}^{\mathcal{P}}_{A,\,Type}[\,]$ can play the role of the **squash, bracket, (-1)-truncation** type constructor, we can take $||\sigma|| \triangleq \mathcal{L}^{\sigma \text{ inhabited}}_{\sigma,\,Type}[\sigma]$ and we get the introduction and elimination rules:

$$\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \mathcal{L}^{\sigma \text{ inhabited}}_{\sigma,\,Type}[M] : \mathcal{L}^{\sigma \text{ inhabited}}_{\sigma,\,Type}[\tau]} \; \mathit{Intro}$$

$$\frac{\Gamma \vdash \lambda x : \sigma.\ M \ : \sigma \to \tau \qquad x \notin FV(M \cup \tau)}{\Gamma \vdash \lambda x : \mathcal{L}^{\sigma \text{ inhabited}}_{\sigma,\,Type}[\sigma].\ M : \mathcal{L}^{\sigma \text{ inhabited}}_{\sigma,\,Type}[\sigma] \to \tau} \; \mathit{Rec} \cdot 1$$

$$\frac{\Gamma \vdash \lambda x : \sigma.\ M \ : \sigma \to \tau \qquad x \notin FV(M \cup \tau)}{\Gamma \vdash \mathcal{L}^{\sigma \text{ inhabited}}_{\sigma,\,Type}[M] : \mathcal{L}^{\sigma \text{ inhabited}}_{\sigma,\,Type}[\tau]} \; \mathit{Rec} \cdot 2$$

$$\frac{\Gamma \vdash \rho : \mathcal{L}^{\sigma \text{inhabited}}_{\sigma,\,Type}[\sigma] \to \ Type \qquad \Gamma, x : \sigma \vdash N : \rho(\mathcal{L}^{\sigma \text{inhabited}}_{\sigma,\,Type}[x])}{\Gamma, x : \mathcal{L}^{\sigma \text{inhabited}}_{\sigma,\,Type}[\sigma] \vdash N : \rho x} \; \mathit{Ind}.$$

# Applications of Locks: Generalized Propositions

- In order to make sense of **partially defined propositions** such as "$x \neq 0 \supset x^{-1} \neq 0$" Martin-Löf's defines $A \supset B$ by

$$\frac{A\ \textbf{Prop} \qquad \begin{array}{c} A\ \textbf{True} \\ B\ \textbf{Prop} \end{array}}{A \supset B\ \textbf{Prop}}$$

using locks we express this by

$$\frac{A\ \ \textbf{Prop} \qquad \mathcal{L}^{A\textbf{True}}_{A,\textbf{Prop}}[B\ \textbf{Prop}]}{A \supset B\ \textbf{Prop}}\ .$$

- **Typical ambiguity** the intended use of $\mathcal{U} \in \mathcal{U}$ can be expressed using locks by

$$\mathcal{L}^{\Phi[\mathcal{U}\in\mathcal{U}]\ \text{is stratifiable}}_{\mathcal{U},Type}[\Phi]$$

# Coq-definitional Implementation of LLF$_{\mathcal{P}}$ - 1

Our goal is twofold: delegating LLF$_{\mathcal{P}}$'s metalanguage to Coq's metalanguage and reducing **inhabitation-search** in LLF$_{\mathcal{P}}$ to **proof-search** in Coq

$$\ulcorner \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \urcorner \quad \leadsto \quad \Pi_{x:\ulcorner \mathcal{P} \urcorner(\ulcorner N \urcorner, \ulcorner \sigma \urcorner)} \ulcorner \rho \urcorner$$

$$\ulcorner \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \urcorner \quad \leadsto \quad \lambda x : \ulcorner \mathcal{P} \urcorner(\ulcorner N \urcorner, \ulcorner \sigma \urcorner) \ulcorner M \urcorner$$

$$\ulcorner \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \urcorner \quad \leadsto \quad \ulcorner M \urcorner x$$

The last encoding is slightly problematic because it is necessary that a *witness* $x : \ulcorner \mathcal{P} \urcorner(\ulcorner N \urcorner, \ulcorner \sigma \urcorner)$ be available. This is available in the case of $O \cdot Top \cdot Unlock$, but in the case of $O \cdot Guarded \cdot Lock$ one needs to refer to the external bound variable. However, in practice, it can be “hidden” using **user defined tactics** in Coq.
The issue would not have arisen if we would have used

$$\ulcorner \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \urcorner \quad \leadsto \quad \ulcorner \textbf{let } x = M \textbf{ in } x \urcorner.$$

# Definitional Implementation of LLF$_\mathcal{P}$ in Coq- 1

- Lock constructor for families

```
Definition lockF := fun s: Set => fun N: s => fun P: s->Prop =>
                    fun r: Prop => forall x: P N, r.
```

- Lock-introduction, i.e. $(O{\cdot}Lock)$ rule

```
Lemma lock: forall s: Set, forall N: s, forall P: s->Prop,
            forall r: Prop, forall M: r, lockF s N P r.
```

- Lock-elimination, i.e. $(O{\cdot}Top{\cdot}Unlock)$ rule

```
Lemma top_unlock: forall s: Set, forall N: s, forall P: s->Prop,
            forall r: Prop, forall M: lockF s N P r, forall x: P N, r.
```

- Guarded lock-elimination, i.e. $(O{\cdot}Guarded{\cdot}Unlock)$ rule

```
Lemma guarded_unlock: forall s: Set, forall S: s, forall P: s->Prop,
            forall t: Prop, forall r: t->Prop,
            forall M: forall y: t, lockF s S P (r y),
            forall N: lockF s S P t, forall x: P S, r (N x).
```

# The Guarded Unlock Tactic

**Ltac**

```
Guarded_unlock x :=
 match goal with
 [ |- lockF _ _ _ ?A ] =>
 unfold lockF;
 apply guarded_unlock with (r := (fun w: x => A));
 [> intro; apply lock | idtac ]
 end.
```

# Implementation of CLLF$_{\mathcal{P}?}$ in Coq

In a paper [MSCS2018] we introduced the system CLLF$_{\mathcal{P}?}$ which allows for the external tool to **synthesize a witness**. The $\mathcal{L}^{\mathcal{P}}_{?x,\sigma}[\,]$ becomes a *binding operator*. The crucial rules are

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma\sigma \vdash \mathcal{L}^{\mathcal{P}}_{?x,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{?x,\sigma}[\tau]} \; O{\cdot}? \cdot Guarded \cdot Lock$$

$$\frac{\Gamma \vdash \mathcal{L}^{\mathcal{P}}_{?x,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{?x,\sigma}[\tau] \qquad \mathcal{P}(\Gamma \vdash N : \sigma)}{\Gamma \vdash M[N/x] : \tau[N/x]} \; O{\cdot}? \cdot Top \cdot Unlock$$

The previous encoding can be accommodated naturally as follows:

$$\ulcorner \mathcal{L}^{\mathcal{P}}_{x,\sigma}[\rho] \urcorner \qquad \leadsto \qquad \Pi_{x:\ulcorner\sigma\urcorner}\Pi_{y:\ulcorner\mathcal{P}(\urcorner(\ulcorner x\urcorner,\ulcorner\sigma\urcorner)}\ulcorner\rho\urcorner$$

# Call-by-value $\lambda$-calculus and its LLF$_{\mathcal{P}}$'s signature $\Sigma_v$

- Syntax of untyped $\lambda$-calculus $\quad M, N ::= x \mid M\ N \mid \lambda x.M$

```
nat: Type       0: nat           free: nat → term
term: Type      S: nat → nat     app: term → term → term
                                 lam: (term → term) → term
```

- Call-by-value equational theory

$$\frac{}{\vdash_{CBV} M = M}\ (\text{refl}) \qquad \frac{\vdash_{CBV} N = M}{\vdash_{CBV} M = N}\ (\text{symm})$$

$$\frac{\vdash_{CBV} M = N \quad \vdash_{CBV} N = P}{\vdash_{CBV} M = P}\ (\text{trans}) \qquad \frac{\vdash_{CBV} M = N \quad \vdash_{CBV} M' = N'}{\vdash_{CBV} MM' = NN'}\ (\text{app})$$

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

where values are either variables or abstractions

eq: term $\to$ term $\to$ Type

eq_app: $\Pi$M,N,P,Q:term. eq M N $\to$ eq P Q $\to$ eq (app M P) (app N Q)

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

csiv: $\Pi$M,N:term $\to$ term.
($\Pi$x:term. $\mathcal{L}^{Val}_{x,term}$[eq (M x) (N x)]) $\to$ eq (lam M) (lam N)

# Call-by-value $\lambda$-calculus in Coq

- Syntax and oracle

```
Parameter term: Set. Parameter lam: (term -> term) -> term.
Parameter free: nat -> term. Parameter app : term -> term -> term.
Definition Val := fun N:term => (exists n, N = (free n))
                                \/
                                (exists M, N = (lam M)).
```

- Equational theory (essential rules)

```
Parameter eq: term -> term -> Prop.
Parameter betav:  forall M:term->term, forall N:term,
                  lockF term N Val (eq (app (lam M) N) (M N)).
Parameter csiv:   forall M N:term->term,
                  (forall x:term, lockF term x Val (eq (M x) (N x)))->
                  eq (lam M) (lam N).
```

# An example proof

Proof of the equation $\lambda x.\ z\ ((\lambda y.y)\ x) = \lambda x.\ z\ x$ via $(O{\cdot}\mathit{Guarded}{\cdot}\mathit{Unlock})$

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{z{:}t \vdash eq(z,z)}\,(\mathit{refl})
\quad
\dfrac{}{x{:}t,\ w{:}eq(app(lam(\lambda y{:}t.\ y),x),x) \vdash eq(app(lam(\lambda y{:}t.\ y),x),x)}\,(\mathit{Hyp})
}{z,\ x{:}t,\ w{:}\ldots \vdash eq(app(z,\ app(lam(\lambda y{:}t.\ y),x)),\ app(z,x))}\,(\mathit{eq\_app})
\quad
\dfrac{}{x{:}t \vdash \mathcal{L}^{Val}_{x,t}[eq(app(lam(\lambda y{:}t.\ y),x),x)]}\,(\mathit{betav})
}{z{:}t \vdash \forall x{:}t.\ \mathcal{L}^{Val}_{x,t}[eq(app(z,\ app(lam(\lambda y{:}t.\ y),x)),\ app(z,x))]}\,(\mathit{GuardedUnlock})
}{z{:}t \vdash eq(\lambda x{:}t.\ app(z,\ app(lam(\lambda y{:}t.\ y),x)),\ \lambda x{:}t.\ app(z,x))}\,(\mathit{csiv})
$$

# The *Fast and Loose and Parallel Reasoning Paradigms*

- A historical example in *Rhind Papyrus* 1650 BC, and *Liber Abaci* by Fibonacci 1200 AD: **regula falsi** for solving linear equations

$$\frac{\mathcal{L}^{f(a)=A}_{f(a)=a,\mathtt{Int}}[M(a)=N] \qquad Kf(a)=A}{M(Ka)=kN}$$

  provided all arithmetical terms are linearand M is homogenous.
- A modern example [DHJG POPL'06]: reasoning correctly on **possibly non-terminating** programming languages assuming that **data are total and finite**;
- A more visionary example in **Quantum Computing**: both in parall execution and in **counterfactual computing**, *i.e.* computing without executing and truth-without-proof .
- Ordinary examples **Fitch-Prawitz Set Theory** or **Typical Ambiguity**,
- **When** are checks performed? **When** are locks removed?
- Monads usually do not allow exiting: but all monads have a **"morally correct" inverse**.
- In the implementation of **Fitch-Prawitz Set Theory** checks are performed when `elim`-rules are applied.
- In LLF$_{\mathcal{P}}$ checks are **implicitly** run in parallel.

# Branch prediction (BP)

In processor architecture, a **branch predictor** is a construct that tries to guess which branch the control will exit, *e.g.* in an `if-then-else`, before the result of the test is actually known, in order to improve the flow in the instruction pipeline.

- **SAFETY PRINCIPLE**: in case of **misprediction** the execution is discarded and resumed starting from the correct branch;
- **MISPREDICTION RECOVERY PROTOCOL**
  - **static** *e.g.* tests or jumps are *never* performed or performed *only* if produce backward-jumps;
  - **dynamic** *i.e.* information is collected at runtime, *e.g.* assume the test is true if it has be true "most" of the times.

We study formally BP for the *Unlimited Register Machine (URM)*:

$$
\begin{array}{llllll}
s & ::= & \langle \iota \mapsto r_\iota \rangle^{\iota \in [0..\infty]} & r_\iota \in \mathbb{N} & \text{Store} \\
I & ::= & Z(i) \mid S(i) \mid T(i,j) \mid J(i,j,k) & i,j,k \in \mathbb{N} & \text{Instruction} \\
P & ::= & (\iota \mapsto I_\iota)^{\iota \in [1..m]} & m \in \mathbb{N} & \text{Program}
\end{array}
$$

whose instructions Zero, Successor, Transfer, Jump have the intended meanings:

$$
\begin{array}{lll}
Z(i) & \triangleq & 0 \rightarrow R_i \\
S(i) & \triangleq & r_i + 1 \rightarrow R_i \\
T(i,j) & \triangleq & r_i \rightarrow R_j \\
J(i,j,k) & \triangleq & \text{if } r_i = r_j \text{ then execute the } k\text{-th instruction else the next one}
\end{array}
$$

# BP: definitions

- Program evaluation

$$E(P,n,s) \quad = \quad \begin{cases} s & \text{if } \mathit{fetch}(P,n){=}\mathit{Halt} \\ E(P,n{+}1,\mathit{zero}(s,i)) & \text{if } \mathit{fetch}(P,n){=}Z(i) \\ \ldots & \ldots \\ E(P,k,s) & \text{if } \mathit{fetch}(P,n){=}J(i,j,k) \text{ and } s(i){=}s(j) \\ E(P,n{+}1,s) & \text{if } \mathit{fetch}(P,n){=}J(i,j,k) \text{ and } s(i){\neq}s(j) \end{cases}$$

- Auxiliary functions

$$\begin{array}{lcl} \mathit{fetch}(P,n) & \triangleq & \text{if } n{>}\mathit{length}(P) \text{ then } \mathit{Halt} \text{ else } I_n \\ \mathit{zero}(s,i) & \triangleq & \lambda\iota{\in}\mathbb{N}.\ \text{if } \iota{=}i \text{ then } 0 \text{ else } s(\iota) \\ \mathit{succ}(s,i) & \triangleq & \lambda\iota{\in}\mathbb{N}.\ \text{if } \iota{=}i \text{ then } s(\iota){+}1 \text{ else } s(\iota) \\ \mathit{move}(s,i,j) & \triangleq & \lambda\iota{\in}\mathbb{N}.\ \text{if } \iota{=}j \text{ then } s(i) \text{ else } s(\iota) \end{array}$$

- LLF$_\mathcal{P}$'s signature for stores and programs

```
nat: Type     0: nat      S: nat → nat
store: Type   zeros: store     cs: nat → store → store
ins: Type     Ht: ins     Zr: nat → ins    ...   Jp: nat → nat → nat → ins
pgm: Type     void: pgm      cp: ins → pgm → pgm
```

# BP: semantics

- Structured evaluation (essential rules)

$$\frac{fetch(P,n){=}Z(i)}{\langle n,s\rangle \rightsquigarrow^{P} \langle n{+}1, zero(s,i)\rangle}\ (\mathrm{eZ}) \qquad \frac{\langle n,s\rangle \rightsquigarrow^{P} \langle m,t\rangle \quad \langle m,t\rangle \rightsquigarrow^{P} \langle q,u\rangle}{\langle n,s\rangle \rightsquigarrow^{P} \langle q,u\rangle}\ (\mathrm{trans})$$

$$\frac{fetch(P,n){=}J(i,j,k) \quad s(i){=}s(j)}{\langle n,s\rangle \rightsquigarrow^{P} \langle k,s\rangle}\ (\mathrm{Jt}) \qquad \frac{fetch(P,n){=}J(i,j,k) \quad s(i){\neq}s(j)}{\langle n,s\rangle \rightsquigarrow^{P} \langle n{+}1,s\rangle}\ (\mathrm{Jf})$$

$$\frac{fetch(P,n){=}Halt}{\langle n,s\rangle \Rightarrow^{P} s}\ (\mathrm{empty}) \qquad \frac{\langle n,s\rangle \rightsquigarrow^{P} \langle m,t\rangle \quad fetch(P,m){=}Halt}{\langle n,s\rangle \Rightarrow^{P} t}\ (\mathrm{stop})$$

- LLF$_{\mathcal{P}}$'s signature for evaluation (*Jump* rules)

$$\begin{array}{lcl}
\mathtt{T} & : & \mathtt{Type} \\
\mathtt{fetch} & : & \mathtt{pgm} \to \mathtt{nat} \to \mathtt{ins} \to \mathtt{Type} \\
\mathtt{step} & : & \mathtt{prg} \to \mathtt{nat} \to \mathtt{store} \to \mathtt{nat} \to \mathtt{store} \to \mathtt{Type} \\
\mathtt{sJt} & : & \Pi\mathtt{P{:}pgm}.\ \Pi\mathtt{n,i,j,k{:}nat}.\ \Pi\mathtt{s{:}store}. \\
 & & \mathtt{fetch\ P\ n\ (J\ i\ j\ k)} \to \mathcal{L}^{\mathtt{Eq}}_{\langle \mathtt{s,i,j}\rangle,\mathtt{T}}[\mathtt{step\ P\ n\ s\ k\ s}] \\
\mathtt{sJf} & : & \Pi\mathtt{P{:}pgm}.\ \Pi\mathtt{n,i,j,k{:}nat}.\ \Pi\mathtt{s{:}store}. \\
 & & \mathtt{fetch\ P\ n\ (J\ i\ j\ k)} \to \mathcal{L}^{\mathtt{Neq}}_{\langle \mathtt{s,i,j}\rangle,\mathtt{T}}[\mathtt{step\ P\ n\ s\ (S\ n)\ s}]
\end{array}$$

# BP: formalization in Coq

- Syntax and oracle

```
Definition store: Set := list nat.
Parameter ins: Set. Parameter Ht: ins. ... Definition pgm: Set := list
Inductive T: Set := triple: store -> nat -> nat -> T.
Definition pr1 (x:T): store := match x with triple s i j => s end. ...
Definition s_nth (s:store) (n:nat): nat := nth n s 0.
Definition Eq := fun x:T => s_nth (pr1 x) (pr2 x) =
                            s_nth (pr1 x) (pr3 x). ...
```

- Semantics (*Jt* rule)

```
Parameter step: pgm -> nat -> store -> nat -> store -> Prop.
Parameter sJt: forall P n i j k s, fetch P n = (Jp i j k) ->
               lockF T (triple s i j) Eq (step P n s k s). ...
```

- A sample proof

$$\dfrac{\dfrac{\dfrac{P(1){=}J(0,1,0)}{\mathcal{L}^{Eq}_{\langle s,0,1\rangle,T}[\langle 1,s\rangle \leadsto^{P} \langle 0,s\rangle]}\,(\mathrm{sJt}) \quad \dfrac{P(0){=}Z(0)}{\langle 0,s\rangle \leadsto^{P} \langle 1,t\rangle}\,(\mathrm{sZ})}{\mathcal{L}^{Eq}_{\langle s,0,1\rangle,T}[\langle 1,s\rangle \leadsto^{P} \langle 1,t\rangle]}\,(\mathrm{sTr}) \qquad Eq(\langle s,0,1\rangle)}{\langle 1,s\rangle \leadsto^{P} \langle 1,t\rangle}\,(\mathrm{O{\cdot}Top})$$

# Adequacy Statements - tentative

- **All** possible executions are adequately modeled in the **"soup"** of provable judgements.
- How are **misprediction recovery protocols** rendered?
- We need to introduce *rules for* **exiting** *or* escaping *from monads*:
- the standard rule is the *Unlock*-rule

$$\frac{\mathcal{L}^{Eq}_{<s,i,j>,T}[V, V'] \qquad Eq(<s,i,j>)}{V}\ standard\ ;$$

- in the **never** protocol we *never* use the rule *sJt* but only *sJf*. The *Unlock*-rule then takes the form

$$\frac{\mathcal{L}^{Neq}_{<s,i,j>,T}[<V, \mathtt{step\ P\ n\ s\ k\ s'}>] \qquad \neg Neq(<s,i,j>)}{\mathtt{step\ P\ n\ s\ k\ s}}\ never\ ;$$

- in the **backwards** protocol a modified *sJt*-rule is used which checks first that the potential jump is a backwards-jump, otherwise the *sJf*-rule is used.
- In the last two examples locked judgements have to record the alternative which originally was not chosen.

# Optimistic concurrency control (OCC)

In information technology, concurrency control ensures that concurrent operations generate correct results, efficiently.

The optimistic approach, in particular, assumes that multiple transactions can be frequently completed without interfering with each other:

- transactions are allowed to use resources without acquiring locks on them
- when a transaction $A$ is completed, it is checked that no other transaction $B$, completed after the activation of $A$, has modified the data that $A$ has used
  - if the check reveals interference, $A$ is rolled back and restarted sequentially
  - otherwise $A$ is allowed to commit its modifications, which are made permanent

Therefore, by assuming that concurrent transactions modify resources just locally before committing, we state that transaction $i$ may perform the following actions:

$$\begin{array}{lcl}
\mathit{start}(i) & \triangleq & \text{activation (first action of the transaction)} \\
\mathit{read}(i,j) & \triangleq & \text{reading on resource } j \\
\mathit{write}(i,j) & \triangleq & \text{writing to resource } j \\
\mathit{check}(i) & \triangleq & \text{check againts interference (last action): commit vs roll back}
\end{array}$$

# OCC: definitions and semantics

- Syntax of schedules

$$
\begin{array}{llllr}
T & \triangleq & \mathbb{N} & & \text{Transaction} \\
R & \triangleq & \mathbb{N} & & \text{Resource} \\
A & ::= & \mathit{start}(i) \mid \mathit{check}(i) \mid & i{\in}T & \text{Action} \\
 & & \mathit{read}(i,j) \mid \mathit{write}(i,j) & j{\in}R & \\
S & ::= & (\iota{\mapsto}A_\iota)^{\iota\in[1..m]} & m{\in}\mathbb{N} & \text{Schedule}
\end{array}
$$

- Datatypes, forming the *state*

$$
\begin{array}{lllr}
\mathit{ctr} & \triangleq & \mathit{stack}(A) & \text{Activation control } (\mathit{begin} \text{ and } \mathit{check} \text{ actions}) \\
\mathit{seq} & \triangleq & T \to \mathit{queue}(A) & \text{Actions of transactions, in sequential order} \\
\mathit{usr} & \triangleq & T \to \mathit{list}(R) & \text{Used resources (by transactions)} \\
\mathit{wrt} & \triangleq & R \to \mathit{list}(T) & \text{Writing transactions (to resources)}
\end{array}
$$

- Semantics, where $M \triangleq \langle \mathit{ctr}, \mathit{seq}, \mathit{usr}, \mathit{wrt} \rangle \in \mathit{state}$ and $C \triangleq \mathbb{N} \times \mathit{state}$

$$
\mathcal{L}^{\mathrm{Opt}}_{\langle i,M\rangle,C}[\{\mathit{check}(i) :: S, M\} \leadsto \{S, \langle \mathit{check}(i) :: \mathit{ctr}, \mathit{seq}'_i, \mathit{usr}, \mathit{wrt}\rangle\}]
$$
$$
\mathcal{L}^{\mathrm{Itf}}_{\langle i,M\rangle,C}[\{\mathit{check}(i) :: S, M\} \leadsto \{S', \langle \mathit{remove}(\mathit{start}(i), \mathit{ctr}), \mathit{seq}'_i, \mathit{usr}'_i, \mathit{delete}(i, \mathit{wrt})\rangle\}]
$$

# OCC: oracle and example

- Predicates

$$\mathrm{Itf}(\Gamma \vdash_\Sigma \langle i, M\rangle : C) \triangleq \exists k{:}T, \exists h{:}R.\ check(k) >_{ctr} start(i) \wedge h \in usr(i) \wedge k \in wrt(h)$$

$$\mathrm{Opt}(\Gamma \vdash_\Sigma \langle i, M\rangle : C) \triangleq \forall k{:}T.\ \neg(check(k) >_{ctr} start(i)) \vee check(k) >_{ctr} start(i) \Rightarrow \forall h{:}R.\ (h \in usr(i) \Rightarrow k \notin wrt(h))$$

- Example: lost update

| Transaction a | Transaction b |
|---|---|
| $start(a),\ read(a, x)$ | |
| | $start(b),\ read(b, x),\ write(a, x),\ check(b)$ |
| $write(a, x),\ check(a)$ | |

It is apparent that check(b) succeeds, whereas check(a) does not, therefore the transaction *a* must be rolled back and processed in sequential mode.

# Conclusion

- We contributed to the development of LLF$_{\mathcal{P}}$
  - giving a definitional implementation in Coq
  - exploring and experimenting with the "fast and loose" paradigm
  - suggesting extensions of the framework and offering insights into implementation.
- We are working on logical combinations of predicates (conjunctions and disjunctions) in locks and their handling via user-defined tactics
- We intend to explore how to prototype an alternate editor for LLF$_{\mathcal{P}}$ using the MMT UniFormal Framework of F. Rabe