Toward a Theory of Contexts of Assumptions in Logical Frameworks

Alberto Momigliano

Department of Computer Science
Università degli Studi di Milano, Italy

Joint work with Amy Felty and Brigitte Pientka
Introduction

Background and motivation

• Logical frameworks: (mechanized) meta-logics for representing, reasoning and programming (over) formal systems.

• Our focus: Specifying formal systems using higher-order abstract syntax (HOAS):
  • binders in the object language ⇐⇒ binders in meta-language.

• The 3 tenets of HOAS:
  • $\alpha$-renaming for free
  • substitution as $\beta$ reduction
  • contexts are implicitly handled via hypothetical parametric judgments

• The first two items are well understood, the third one somewhat less

This talk: let’s revisit reasoning with assumptions/open objects
Hypothetical judgments are well known and non controversial as far as \textit{representation} is concerned since the late 80's (\lambda Prolog, Elf, Isabelle Pure)

Less consensus on the \textit{reasoning} side, different approaches along the Type/Proof Theory divide (and among TT as well . . .)

\begin{tabular}{l|l}
Type-theorists & Proof-theorists \\
\hline
Twelf, Beluga, Delfin . . . & Abella, Tac, Hybrid . . .
\end{tabular}

An analogy: In the beginning, Gentzen created natural deduction, but then he switched to the sequent calculus in order to sort out the meta-theory.
An Homage to ProofCert

- We all want to relate one framework to another with the aim to transfer theorems and proofs.

- There is ongoing work on relating TT and PT logical frameworks, mainly Minneapolis-based:
  - LF to $\lambda$Prolog
  - Twelf to $M_2$ (only closed terms, so Twelf 1.2)

- ...but issue of transferring reasoning in presence of assumptions is still unaddressed, e.g.
  * What is the logical status of Twelf’s *regular world assumption*?
  * How do you map Beluga’s contextual objects to a logic such as $G$ or Coq?
  * ...
The rest of the talk

• Motivating examples

• Notation for contexts - not just a matter of style.
  • Contexts as structured sequences
  • Generalized contexts
  • Context relations

• Some very preliminary remarks about:
  • A unifying view of generalized context and ctx relation via the lattice of context assumptions;
  • the design of ORBI (Open challenge problem Repository for systems supporting reasoning with BInders), an intermediate language for specifying benchmarks problems.
A first example: the polymorphic lambda-calculus

Grammar: Types and Terms - does not enforce scope

Types $T ::= \alpha$

Terms $M ::= \lambda x : T . M \mid \text{arr} T_1 T_2 \mid \text{all} \alpha . T \mid \text{app} M N \mid \text{tlam} \alpha . M \mid \text{tapp} M T$
A first example: the polymorphic lambda-calculus

Grammar: Types and Terms - does not enforce scope

Types \( T \) ::=
\[ \alpha \]
| \( \text{arr} \, T_1 \, T_2 \)
| \( \text{all} \, \alpha. \, T \)

Terms \( M \) ::=
\[ x \]
| \( \text{lam} \, x: \, T \, . \, M \)
| \( \text{app} \, M \, N \)
| \( \text{tlam} \, \alpha. \, M \)
| \( \text{tapp} \, M \, T \)

Alternative: Well-formed terms Martin-Löf-style - enforces scope

\[ \frac{x \text{ term}}{tm_v} \]
\[ \frac{M \text{ term}}{(\text{lam} \, x. \, M) \text{ term}} \]
\[ \frac{M_1 \text{ term}}{(\text{app} \, M_1 \, M_2) \text{ term}} \]

\[ \frac{\alpha \text{ tp}}{tp_v} \]
\[ \frac{M \text{ term}}{(\text{tlam} \, \alpha. \, M) \text{ term}} \]
\[ \frac{M \text{ term}}{(\text{tapp} \, M \, A) \text{ term}} \]
More examples in context-free representation

Another example: some rules for “algorithmic” equality (the copy clause)

\[
\begin{align*}
\frac{x \text{ term} \quad x \quad aeq \ x \ x}{aeq} \\
\frac{aeq \ M \ N}{aeq \ (\text{lam} \ x. \ M) \ (\text{lam} \ x. \ N)} \quad ae^{-x,ae}
\end{align*}
\]

\[
\frac{aeq \ M_1 \ N_1 \quad aeq \ M_2 \ N_2}{aeq \ (\text{app} \ M_1 \ M_2) \ (\text{app} \ N_1 \ N_2)} \quad ae_a
\]

- Context-free representation scales from grammars to judgments
- 2-dimensional notation is ambiguous
- Can we tell open vs. closed object?
- What about structural properties of assumptions? Shouldn’t they be explicit?
What is a context?
What is a context?

- Typical answer [From Gentzen on]: A sequence – OK, sometimes a (multi)set – of formulas $A_1, A_2, \ldots, A_n$. 

Issue: The use of ‘,’ is ambiguous.

Our view: Contexts are structured sequences - distinguish between “blocks” and ctx using ‘;’ and ‘,’. This was already adopted in Twelf 1.3 with the notion of regular world.
Putting things into context

What is a context?

- Typical answer [From Gentzen on]: A sequence – OK, sometimes a (multi)set – of formulas $A_1, A_2, \ldots, A_n$.

Examples of contexts occurring in the above examples:

- Type Context $\Gamma ::= \cdot \mid \Gamma, \alpha \text{ tp}$
- Term/Type Context $\Gamma ::= \cdot \mid \Gamma, \alpha \text{ tp} \mid \Gamma, x \text{ term}$
- Eq. Context $\Gamma ::= \cdot \mid \Gamma, x \text{ term}, \text{aeq } x x$  
  We are introducing the variable $x$ together with the assumption $\text{aeq } x x$
Putting things into context

What is a context?

- Typical answer [From Gentzen on]: A sequence – OK, sometimes a (multi)set – of formulas $A_1, A_2, \ldots, A_n$.

Examples of contexts occurring in the above examples:

Type Context $\Gamma$ ::= · | $\Gamma, \alpha$ tp

Term/Type Context $\Gamma$ ::= · | $\Gamma, \alpha$ tp | $\Gamma, x$ term

Eq. Context $\Gamma$ ::= · | $\Gamma, x$ term, aeq $x$ $x$

We are introducing the variable $x$ 

together with the assumption aeq $x$ $x$

Issue: The use of ‘,’ is ambiguous.

Our view: Contexts are structured sequences - distinguish between “blocks” and ctx using ‘;’ and ‘,’

This was already adopted in Twelf 1.3 with the notion of regular world
A context is a sequence of declarations $D$ where a declaration is a block of individual atomic assumptions separated by ';', which binds tighter than ',',

Atom $A$
Block of declaration $D ::= A | D; A$
Context $\Gamma ::= \cdot | \Gamma, D$
Schema $S ::= D_s | D_s | S$
Contexts as structured sequences

- A context is a sequence of declarations $D$ where a declaration is a block of individual atomic assumptions separated by ';', which binds tighter than ','.

\[
\begin{align*}
\text{Atom} & : A \\
\text{Block of declaration} & : D ::= A | D; A \\
\text{Context} & : \Gamma ::= \cdot | \Gamma, D \\
\text{Schema} & : S ::= D_s | D_s | S
\end{align*}
\]

- A schema classify a context, where ' | ' describes alternatives

\[
\begin{align*}
S_{\alpha x} & ::= \alpha \, \text{tp} | x \, \text{term} \\
S_{x ae q} & ::= x \, \text{term}; aeq \, x \, x
\end{align*}
\]

- There are some obvious typing rules relating context and schemas, not shown here.

- **Convention:** $\Phi_{\alpha x}$ describes a context with schema $S_{\alpha x}$. 
Polymorphic lambda-calculus - revisited (with explicit context)

Well-formed Terms

\[
\begin{align*}
\Phi_{\alpha x}, \alpha \text{ tp} & \vdash M \text{ term} \quad \Phi_{\alpha x} \vdash (\text{tapp} M A) \text{ term} & \text{tm}_{ta} \\
\Phi_{\alpha x} \vdash (\text{app} M_1 M_2) \text{ term} & \Phi_{\alpha x} \vdash M_1 \text{ term} \Phi_{\alpha x} \vdash M_2 \text{ term} & \text{tm}_{a} \\
\Phi_{\alpha x}, \alpha \text{ tp} & \vdash M \text{ term} \quad \Phi_{\alpha x} \vdash (t\lambda \alpha. M) \text{ term} & \text{tm}_{tl} \\
\Phi_{\alpha x} \vdash (t\lambda \alpha. M) \text{ term} & \Phi_{\alpha x} \vdash \text{tlam} \alpha. M \text{ term} \quad \Phi_{\alpha x} \vdash \text{tlam} \alpha. M \text{ term} \quad \Phi_{\alpha x} \vdash \text{tlam} \alpha. M \text{ term} & \text{tm}_{l} \\
\Phi_{\alpha x} \vdash \text{tlam} \alpha. M \text{ term} & \Phi_{\alpha x} \vdash \text{tlam} \alpha. M \text{ term} \quad \Phi_{\alpha x} \vdash \text{tlam} \alpha. M \text{ term} & \text{tm}_{l}
\end{align*}
\]

\[
\begin{align*}
\Phi_{\alpha x} & \vdash \alpha \text{ tm} \quad \Phi_{\alpha x} \vdash M \text{ term} \quad \Phi_{\alpha x} \vdash \text{lam} x. M \text{ term} & \text{tm}_{v} \\
\Phi_{\alpha x} \vdash \text{lam} x. M \text{ term} & \Phi_{\alpha x}, \alpha \text{ tm} \vdash M \text{ term} \quad \Phi_{\alpha x}, \alpha \text{ tm} \vdash \text{lam} x. M \text{ term} & \text{tm}_{v}
\end{align*}
\]
Structural rules

- More fine-grained view of structural rules, which can be applied inside a block or to a whole ctx;
- Slightly unusual presentation of rules based on operations on declarations:
  - Let \( \text{rm}_A : S \rightarrow S' \) be a total function taking \( D \in S \) and returning \( D' \in S' \) where \( D' \) is \( D \) with \( A \) removed, if \( A \in D \); otherwise \( D' = D \).
  - Let \( \text{perm}_\pi : S \rightarrow S' \) be a total function which permutes the elements of \( D \in S \) according to \( \pi \) to obtain \( D' \in S' \).
- Note that we also “remove” whole declarations (\( \text{rm}_D \)).
- This approach will hopefully pay off soon enough...
Structural properties of declarations

- **Declaration Weakening:**
  \[
  \frac{\Gamma, \text{rm}_A(D), \Gamma' \vdash J}{\Gamma, D, \Gamma' \vdash J} \quad d-wk
  \]

- **Declaration Strengthening:**
  \[
  \frac{\Gamma, D, \Gamma' \vdash J}{\Gamma, \text{rm}_A(D), \Gamma' \vdash J} \quad d-str(\dagger)
  \]
  with the proviso (\dagger) that \( A \) is irrelevant to \( J \) (read *subordination*).

- **Declaration Exchange**
  \[
  \frac{\Gamma, D, \Gamma' \vdash J}{\Gamma, \text{perm}_\pi(D), \Gamma' \vdash J} \quad d-exc
  \]
Structural properties of contexts

We canonically extended those operations to act on contexts \( (\text{rm}_A^*, \text{perm}_{\pi}^*) \):

- **Context weakening**
  \[
  \dfrac{\text{rm}_A^*(\Gamma) \vdash J}{\Gamma \vdash J} \quad \text{c-wk}
  \]

- **Context strengthening**
  \[
  \dfrac{\Gamma \vdash J}{\text{rm}_A^*(\Gamma) \vdash J} \quad \text{c-str}(\uparrow)
  \]

- **Context exchange**
  \[
  \dfrac{\Gamma \vdash J}{\text{perm}_{\pi}^*(\Gamma) \vdash J} \quad \text{c-exc}
  \]
Let’s look back at the rule for well formed type application

\[
\frac{\Phi_{\alpha x} \vdash M \text{ term} \quad ?? \vdash A \text{ tp}}{\Phi_{\alpha x} \vdash (\text{tapp } M A) \text{ term}} \quad tm_{ta}
\]

where \( \Phi_{\alpha x} := \cdot \mid \Phi_{\alpha x}, x \text{ term} \mid \Phi_{\alpha x}, \alpha \text{ tp} \)
Examples - revisited

Let’s look back at the rule for well-formed type application

\[
\frac{\Phi_{\alpha x} \vdash M \text{ term } \quad ?? \vdash A \text{ tp}}{\Phi_{\alpha x} \vdash (\text{tapp } M A) \text{ term}} \quad \text{tm}_{ta}
\]

where \( \Phi_{\alpha x} := \cdot | \Phi_{\alpha x}, x \text{ term } | \Phi_{\alpha x}, \alpha \text{ tp} \)

In what context is \( A \) a well-formed type?
Examples - revisited

Let’s look back at the rule for well formed type application

\[
\frac{\Phi_{\alpha x} \vdash M \text{ term} \quad ?? \vdash A \text{ tp}}{\Phi_{\alpha x} \vdash (\text{tapp } M \ A) \text{ term}}
\]

where \( \Phi_{\alpha x} := \cdot \mid \Phi_{\alpha x}, x \text{ term} \mid \Phi_{\alpha x}, \alpha \text{ tp} \)

In what context is \( A \) a well-formed type?

1. \( A \) is a type in \( \Phi_{\alpha x} \), i.e., via implicit weakening: we use one joint context
Introduction

Examples - revisited

Let’s look back at the rule for well formed type application

\[
\frac{\Phi_{\alpha x} \vdash M \text{ term} \quad ?? \vdash A \text{ tp}}{\Phi_{\alpha x} \vdash \text{tapp}(M, A) \text{ term}}
\]

where \( \Phi_{\alpha x} := \cdot | \Phi_{\alpha x}, x \text{ term} | \Phi_{\alpha x}, \alpha \text{ tp} \)

In what context is \( A \) a well-formed type?

1. \( A \) is a type in \( \Phi_{\alpha x} \), i.e., via implicit weakening: we use one joint context

2. \( A \) is a type in \( \text{rm}^*_x \text{ term}(\Phi_{\alpha x}) \): smallest context necessary, explicit strengthening.
Examples - revisited

Let’s look back at the rule for well formed type application

\[
\frac{\Phi_{\alpha x} \vdash M \text{ term} \quad ?? \vdash A \text{ tp}}{\Phi_{\alpha x} \vdash (\text{tapp } M A) \text{ term}} \quad tm_{ta}
\]

where \( \Phi_{\alpha x} := \cdot \mid \Phi_{\alpha x}, x \text{ term} \mid \Phi_{\alpha x}, \alpha \text{ tp} \)

In what context is \( A \) a well-formed type?

1. \( A \) is a type in \( \Phi_{\alpha x} \), i.e., via implicit weakening: we use one joint context

2. \( A \) is a type in \( \text{rm}_{x}^{*} \text{ term}(\Phi_{\alpha x}) \): smallest context necessary, explicit strengthening.

Why do we care?
Examples - revisited

Let’s look back at the rule for well formed type application

\[
\frac{\Phi_{\alpha x} \vdash M \text{ term} \quad ?? \vdash A \text{ tp}}{\Phi_{\alpha x} \vdash (tapp \ M \ A) \text{ term}} \quad tm_{ta}
\]

where \( \Phi_{\alpha x} := \cdot \mid \Phi_{\alpha x}, x \text{ term} \mid \Phi_{\alpha x}, \alpha \text{ tp} \)

In what context is \( A \) a well-formed type?

1. \( A \) is a type in \( \Phi_{\alpha x} \), i.e., via implicit weakening: we use one joint context

2. \( A \) is a type in \( rm_x^* \text{ term}(\Phi_{\alpha x}) \): smallest context necessary, explicit strengthening.

Why do we care?
It’s the meta-theory, stupid!
Reasoning in contexts

Attempt (Admissibility of Reflexivity)

*For every term* $M$, ??? $\vdash$ aeq $M$ $M$.

The proof should be by induction on $M$…

Two possible approaches to fill that ???

1. **Generalized context** approach (G). The context used in the theorem contains all assumptions in the relevant judgments. 
   Think Twelf/Beluga
Introduction

Reasoning in contexts

Attempt (Admissibility of Reflexivity)

For every term $M$, $\vdash aeq M M$.

The proof should be by induction on $M$...

Two possible approaches to fill that $\vdots$

1. Generalized context approach (G). The context used in the theorem contains all assumptions in the relevant judgments. Think Twelf/Beluga

2. Context relations approach (R). State how the different relevant contexts are related (using rm*) and then state the theorem under the condition that the relation holds. Think Abella/Hybrid
Generalized context : Reflexivity proof

- Here the *generalized* context has schema \( x \) term; aeq \( x \) \( x \), so it’s just \( \Phi_{xaeq} \), but in general contains all the relevant assumptions from the contributing contexts.

**Theorem**

If \( \Phi_{xaeq} \vdash M \) term then \( \Phi_{xaeq} \vdash aeq \ M \ M \).
Generalized context : Reflexivity proof

Here the generalized context has schema $x$ term; aeq $x$ $x$, so it's just $\Phi_{xaeq}$, but in general contains all the relevant assumptions from the contributing contexts.

Theorem

If $\Phi_{xaeq} \vdash M$ term then $\Phi_{xaeq} \vdash aeq M M$.

Proof.

\[
\text{rec ref : } \{\phi:xaeqC}\{M:[\phi. \text{ term}]\} \ [\phi. \text{ aeq (M ... ) (M ... )}] = \\
\text{mlam } \phi \Rightarrow \text{mlam } M \Rightarrow \text{case } [\phi. \text{ M ... }] \text{ of } \\
| \ [\phi. \text{ #p.1 ... }] \Rightarrow [\phi. \text{ #p.2 ... }] & \% \text{ Variable} \\
| \ [\phi. \text{ lam } \lambda x. \text{ M ... } x] \Rightarrow & \% \text{ Lambda} \\
\quad \text{let } [\phi, b: \text{block} \ y: \text{term, ae_v: aeq y y. D ... b.1 b.2}]= \\
\quad \text{ref } [\phi, b: \text{block} \ y: \text{term, ae_v: aeq y y}] [\phi, b. \text{ M ... b.1}] \\
\quad \text{in } [\phi. \text{ ae_l } \lambda x. \lambda w. (D ... x w)] \\
| \ [\phi. \text{ app (M1 ...) (M2 ...)]} \Rightarrow & \% \text{ Application} \\
\quad \text{let } [\phi. \text{ D1 ... }] = \text{ref } [\phi] [\phi. \text{ M1 ... }] \text{ in} \\
\quad \text{let } [\phi. \text{ D2 ... }] = \text{ref } [\phi] [\phi. \text{ M2 ... }] \text{ in } [\phi. \text{ ae_a (D1 ...) (D2 ...)]};
\]
Context relations: Reflexivity

- Note that $\Phi_x = \text{rm}_{aeq}^* x(\Phi_{xaeq})$. We can define the graph of this function inductively:

\[
\begin{align*}
\Phi_x \sim & \Phi_{xaeq} \\
\Phi_x, x \text{ term} \sim & \Phi_{xaeq}, x \text{ term}; \text{aeq} x x
\end{align*}
\]

Theorem

Assume $\Phi_x \sim \Phi_{xaeq}$. If $\Phi_x \vdash M$ term then $\Phi_{xaeq} \vdash \text{aeq} M M$. 
Introduction

Context relations: Reflexivity

- Note that $\Phi_x = \text{rm}_{\text{aeq}}^* x (\Phi_{\text{xeq}})$. We can define the graph of this function inductively:

  $\Phi_x \sim \Phi_{\text{xeq}}$

  $\Phi_x, x \text{ term} \sim \Phi_{\text{xeq}}, x \text{ term}; \text{aeq} x x$

  $\text{crel}_{\text{xeq}}$

Theorem

Assume $\Phi_x \sim \Phi_{\text{xeq}}$. If $\Phi_x \vdash M \text{ term}$ then $\Phi_{\text{xeq}} \vdash \text{aeq} M M$.

- In Abella (and Hybrid, give or take), the relations and the statement of the thm look like that:

Define $\text{xeqR} : \text{olist} \rightarrow \text{olist} \rightarrow \text{prop}$ by

  $\text{xeqR nil nil;}$

  $\text{nabla x, xeqR (term x :: Ts) (aeq x x :: As) := xeqR Ts As.}$

Theorem reflR: forall Ts As M, $\text{xeqR Ts As} \rightarrow$

  $\{\text{Ts \vdash term M}\} \rightarrow \{\text{As \vdash aeq M M}\}.$
We extend the previous example (algorithmic equality of terms) by considering *declarative* equality (which adds rules for reflexivity, symmetry, and transitivity), and prove them equivalent.

\[
\begin{align*}
\Phi_{\text{xaeq}} &:= \cdot \mid \Phi_{\text{xaeq}}, x \text{ term}; \text{aeq} x x \text{ (as seen before)} \\
\Phi_{\text{xdeq}} &:= \cdot \mid \Phi_{\text{xdeq}}, x \text{ term}; \text{deq} x x \\
\Phi_{\text{xda}} &:= \cdot \mid \Phi_{\text{xda}}, x \text{ term}; \text{deq} x x; \text{aeq} x x
\end{align*}
\]

Recall the statement of reflexivity for terms:

*If* \( \Phi_{\text{xaeq}} \vdash M \text{ term} *then* \( \Phi_{\text{xaeq}} \vdash \text{aeq} M M.\)

This lemma (and others) are needed in the proof of equivalence, but we must “promote” it first to the larger context \( \Phi_{\text{xda}}.\)

*If* \( \Phi_{\text{xda}} \vdash M \text{ term} *then* \( \Phi_{\text{xda}} \vdash \text{aeq} M M.\)
Lemma

If $\Phi_{xda} \vdash M$ term then $\Phi_{xda} \vdash \text{aeq } M M$.

Proof.

$\Phi_{xda} \vdash M$ term by assumption
$\Phi_{xaeq} \vdash M$ term by \textit{c-str}
$\Phi_{xaeq} \vdash \text{aeq } M M$ by previous lemma
$\Phi_{xda} \vdash \text{aeq } M M$ by \textit{c-wk}

- In general, proofs of promotion for G versions of theorems require a combination of strengthening and weakening on contexts.
- R versions of promotion involve strengthening and weakening of one or both sides of a context relation.
- Wouldn’t it be nice to have your logical framework support this?
The semi-lattice of declarations

- Consider the set $\mathcal{D}$ of well-formed declarations and quotient it under the perm operation;
- Define $D_1 \preceq D_2$ iff there is $D$ s.t. $D_1 = \text{rm}_D(D_2)$, modulo perm, which will ignore from now on.
- Define $D_1 \lor D_2$ as $\text{remove}_\text{dup}(D_1; D_2)$. Hence, “consing” (and cleaning up) two declarations yields their least upper bound.
- $\langle \mathcal{D}, \preceq, \epsilon, \lor \rangle$ is an upper semi-lattice with the empty declaration $\epsilon$ as zero element.
- Extend this construction to the set of schemas$^1$ and of well-formed ctx.

\textit{the generalized context in the G-version of thms can be seen as the lub of the relevant ctx}, e.g.

$$\Phi_{xda} = \Phi_{xaeq} \lor \Phi_{xdeq}$$

\footnote{Warning: we haven’t worked out the details for alternatives yet.}
The lattice 2

- A picture: see white board
The lattice 2

- A picture: see white board
- Phrasing weak/stren as “casting” – remember the promotion lemma:

\[
\frac{\text{rm}^*_D(\Gamma) \vdash J}{\Gamma \vdash J} \quad \text{c-wk} \quad \sim \quad \frac{\Gamma' \vdash J \Gamma \leq^c \Gamma}{\Gamma \vdash J} \quad \text{upc}
\]

\[
\frac{\Gamma \vdash J}{\text{rm}^*_D(\Gamma) \vdash J} \quad \text{c-str} \quad \sim \quad \frac{\Gamma' \vdash J \Gamma \leq^c \Gamma'}{\Gamma \vdash J} \quad \text{doc}
\]

- What about ctx relations? The intuition is that we can recover the certain ctx relations by navigating the Hasse diagram.
- We conjecture that if you give us the G version of a thm involving a ctx Φ, we can recover the R version by relating the ctxs of which Φ is the lub.

Given \( \Phi_{xda} = \Phi_{xaeq} \lor \Phi_{xdeq} \), build \( \Phi_{xaeq} \sim \Phi_{xdeq} \)
• We are designing Open challenge problem Repository for systems supporting reasoning with Blinders, for sharing HOAS benchmark problems – Think an intermediate language between OTT and TPTP

• Uses a Beluga-like syntax enriched with *directives* so that the ORBI2X tools will compile it into legal Twelf/Beluga, Abella/Hybrid etc.

%Syntax
tm: type. app: tm → tm → tm. lam: (tm → tm) → tm.

%Judgments
aeq: tm → tm → type.

%Rules
ae_l: (∑x:tm) aeq x x → aeq (M x) (N x)) → aeq (lam (λx. M x)) (lam (λx. N x))

%Schemas
schema xaeqG: block (x:tm; u:aeq x x).
schema xaeqR: block (x:tm) ~ block (x:tm; u:aeq x x).

%Theorems
theorem reflG : forall (Phi : xaeqG) (M : tm), [Phi |- aeq M M].
%PT explicit (M : tm) in reflG.
Conclusions and future work

What started as a comparison work between HOAS systems is bearing additional fruits:

• A re-appraisal of the role of ctx in Proof Theory – in other terms a percolation of Beluga’s type theory into PT, hopefully not scaring people away;
• The basis of a possible unification of how ctx are mechanized in TT and PT tools
• The design of an intermediate language for benchmark sharing

Current and future work:

• Carry out the G-to-R translation
• Work out the machinery to automate promotion lemmas
Thank you!