Equivalence of System F and $\lambda 2$ in Abella

Jonas Kaiser  Gert Smolka

TTT 2017, Paris

January 15, 2017
The Problem

- Everybody defines their own version.
- *Equivalence* is tacitly assumed when external results are applied.

We consider two particular versions

- $F$: two-sorted, explicit, separate type variable context, e.g. [Harper '13]
- $\lambda_2$: single-sorted pure type system (PTS) [Barendregt '91]

Are the two presentations equivalent? In what sense?

\[ F \approx \lambda_2 \]
Overview

1 Motivation

2 Preliminaries

3 Abella Proof – HOAS

4 Comparison with Coq Proof – de Bruijn
Historical Context

“To show that the two representations of these systems are in fact the same requires some technical but not difficult work.”

Herman Geuvers, in Logics and Type Systems, ’93

- Mostly discusses PTSs.
- “Traditional” systems not defined precisely.
- Desired correspondence only stated, not proven.
- We want a formal/mechanised proof of:

\[
\begin{align*}
&\vdash_{F} s : A \iff \vdash_{2} s' : A' \\
&\vdash_{2} a : b \iff \vdash_{F} a' : b'
\end{align*}
\]

The technical part involves dealing with syntax and variable binding.
Dealing with Syntax

- Binding of variables (of different kinds).
- Free variables vs. bound identifiers; freshness.
- Capture-avoiding substitution(s).

Various approaches:

- HOAS
- LN
- de Bruijn

Increasing level of abstraction.

Requires suitable function spaces.
Abstraction Layer not stable.
Library support is essential.
Reduction of the Typing Problem – 2 Proofs

HOAS proof using syntax relations in Abella:

\[ \{ s : F \ A \} \iff \exists ab. \ \{ s \approx a \} \land \{ A \sim b \} \land \{ a :_2 b \} \]

\[ \{ a :_2 b \} \iff \exists sA. \ \{ s \approx a \} \land \{ A \sim b \} \land \{ s : F \ A \} \]

De Bruijn proof using translation functions in Coq [K/Tebbi/Smolka CPP’17]:

\[ \vdash_F s : A \iff \vdash_2 \llbracket s \rrbracket : \llbracket A \rrbracket \]

\[ \vdash_2 a : b \iff \vdash_F \llbracket a \rrbracket : \llbracket b \rrbracket \]

Note: Utilises the Autosubst de Bruijn library [Schäfer/Tebbi/Smolka '15].
Interactive theorem prover with two layers (two-level logic approach).

Meta level: $G$
- Intuitionistic, predicative fragment of Church’s STT,
- + (co-)inductive predicates,
- + built-in natural numbers and natural induction,
- + nominal quantification ($\forall x.s$): $x$ in $s$ is guaranteed fresh
- Note: no induction on types, no functions.

Specification level: Hereditary Harrop Formulas / $\lambda$Prolog
- Horn clauses (cf. Prolog): $A \dashv:: C, D$
- + hypothetical reasoning: $A \dashv:: C, E \Rightarrow D$
- + quantification: $A \dashv:: C, \forall x. D x$

Logical Embedding:
- HHOP-derivations are inductive
- $\{J\}$ holds in $G \iff J$ has a $\lambda$Prolog derivation
- $\{L \vdash J\}$ holds in $G \iff J$ has a derivation, given hypotheses $L$
### HOAS Signatures of System F and \( \lambda2 \)

<table>
<thead>
<tr>
<th>( \text{Ty}_F, \text{Tm}_F ) type</th>
<th>( \text{Tm}_2 ) type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( _ \rightarrow _ )</td>
<td>( \text{Ty}_F \rightarrow \text{Ty}_F \rightarrow \text{Ty}_F )</td>
</tr>
<tr>
<td>( \forall _ )</td>
<td>( (\text{Ty}_F \rightarrow \text{Ty}_F) \rightarrow \text{Ty}_F )</td>
</tr>
<tr>
<td>( _ \otimes _ )</td>
<td>( \text{Tm}_F \rightarrow \text{Tm}_F \rightarrow \text{Tm}_F )</td>
</tr>
<tr>
<td>( _ \odot _ )</td>
<td>( \text{Tm}_F \rightarrow \text{Ty}_F \rightarrow \text{Tm}_F )</td>
</tr>
<tr>
<td>( \lambda _ _ )</td>
<td>( \text{Ty}_F \rightarrow (\text{Tm}_F \rightarrow \text{Tm}_F) \rightarrow \text{Tm}_F )</td>
</tr>
<tr>
<td>( \Lambda _ )</td>
<td>( (\text{Ty}_F \rightarrow \text{Tm}_F) \rightarrow \text{Tm}_F )</td>
</tr>
</tbody>
</table>
Proof in Abella – Syntax Relations

_ ∼ _ \( \text{Ty}_F \rightarrow \text{Tm}_2 \rightarrow o \)

_ ≈ _ \( \text{Tm}_F \rightarrow \text{Tm}_2 \rightarrow o \)

\[
A \sim a \quad \Pi x. \ B \sim b \ x
\]

\[
A \rightarrow B \sim \Pi a. b
\]

\[
\Pi x \ y. \ x \sim y \ \Rightarrow \ A x \sim a y
\]

\[
\forall. A \sim \Pi^*. a
\]

\[
s \approx a \quad t \approx b
\]

\[
s @ t \approx a @ b
\]

\[
A \sim a \quad \Pi x \ y. \ x \approx y \ \Rightarrow \ s x \approx b y
\]

\[
\lambda A. s \approx \lambda a. b
\]

\[
\Pi x \ y. \ x \sim y \ \Rightarrow \ s x \approx b y
\]

\[
\Lambda. s \approx \lambda^*. b
\]
Proof in Abella, ctnd.

- Show that $\sim, \approx$ are both injective and functional.
- Show that $\sim, \approx$ are conditionally left- & right-total and preserve judgements, e.g. for $\sim$:

\[
\{A \; ty\} \Rightarrow \exists a. \; \{A \sim a\} \land \{a : 2 *\} \\
\{a : 2 *\} \Rightarrow \exists A. \; \{A \sim a\} \land \{A \; ty\}
\]

- Due to injectivity and functionality, witnesses are unique.
- Thus both inverse implications ($\iff$) also hold.
- Similar for $\approx$ but more verbose.
We have to generalise to open terms and hypothetical contexts.

Consider functionality for $\sim$:

$$C_R(L) \Rightarrow \{ L \vdash A \sim a \} \Rightarrow \{ L \vdash A \sim b \} \Rightarrow a = b \quad L : o \ list$$

Inductive definition of $C_R(L)$ using nominals:

$$
\begin{align*}
C_R(\bullet) & \quad C_R(L) \quad x, y \text{ fresh for } L \quad C_R(L) \quad x, y \text{ fresh for } L \\
\hline
& C_R(L, x \sim y) & C_R(L, x \approx y)
\end{align*}
$$

Define $C_R : o \ list \rightarrow prop$ by

$$
\begin{align*}
C_R(\bullet); \\
\nabla x y, \ C_R(L, x \sim y) & := C_R(L); \\
\nabla x y, \ C_R(L, x \approx y) & := C_R(L).
\end{align*}
$$
Contexts, ctnd.

- For \textit{totality/preservation} the situation is similar, e.g.

\[
\{L_F \vdash A \text{ ty}\} \Rightarrow \forall L_R L_2. \quad C(L_F \mid L_R \mid L_2) \Rightarrow \\
\exists a. \quad \{L_R \vdash A \sim a\} \land \{L_2 \vdash a :_2 \ast\}
\]

- \(C(L_F \mid L_R \mid L_2)\) is interesting:
  - 
  - \(\text{Entails } C_F(L_F), \ C_R(L_R)\) and \(C_2(L_2)\) by construction.
  - Recall that \(L_R\) is effectively a \textit{relation on type- and term-variables}.
  - \(\text{Ensures that } L_R\) precisely relates the typing contexts \(L_F\) and \(L_2\).

- Inductive definition:

\[
\begin{array}{c}
C(\bullet \mid \bullet \mid \bullet) \\
\hline
C(L_F \mid L_R \mid L_2) \quad x, y \text{ fresh for } L_F, L_R, L_2
\end{array}
\quad
\begin{array}{c}
C(L_F, x \begin{array}{c}
\text{ty}
\end{array} \mid L_R, x \sim y \mid L_2, y :_2 \ast)
\end{array}
\]

\[
\begin{array}{c}
\{L_F \vdash A \text{ ty}\} \\
\{L_R \vdash A \sim a\} \\
\{L_2 \vdash a :_2 \ast\}
\end{array}
\quad
\begin{array}{c}
C(L_F \mid L_R \mid L_2) \\
\hline
C(L_F, x :_F A \mid L_R, x \sim y \mid L_2, y :_2 a)
\end{array}
\]
Proof relies heavily on useful *inversion lemmas*. Reason:

- Our contexts only contain variable information.
- But every simple case analysis on \( \{ L \vdash J \} \) considers the case \( J \in L \), even if \( J \) is a non-variable judgement.

Applications of \( \lambda 2 \) are particularly involved:

\[
\begin{align*}
\{ L_R \vdash s \approx a \circ b \} & \quad \rightarrow \quad \{ L_R \vdash s' \circ B \approx a \circ b \} \\
\{ L_R \vdash s' \circ B \approx a \circ b \} & \quad \rightarrow \quad \{ L_R \vdash s' \circ t \approx a \circ b \}
\end{align*}
\]

Solving this solely from typing information for \( b \) under some \( L_2 \) appears to rely on the predicate \( C \) to connect \( L_R \) and \( L_2 \). Only having \( C_R(L_R) \) and \( C_2(L_2) \) is not enough.
Comparison with Coq Proof – de Bruijn

Similarities of both proofs

- Overall proof structure.
- *Propagation/Type Correctness* plays a major role.
- The (⇐)-directions are obtained from the respective other (⇒)-result.
- Hardest case: disambiguation of PTS applications.

Differences

- Relations avoid cancellation laws (about a third of the Coq proof).
- The de Bruijn proof clearly separates type formation from typing, in the HOAS proof they are connected much closer.

Main Observation

- The predicate \( C(L_F | L_R | L_2) \) appears to be the *relational combination* of all four de Bruijn morphism conditions. The latter express that certain renaming functions map variable typings from one context to another.
Remarks on Abella Usability

- Overall experience of working in Abella was quite pleasant.
- The combination of the two-level approach and nominals was particularly useful.
- Proof scripts are extremely fragile when it comes to refactoring.
  - Automatically named, but explicitly referenced hypotheses.
  - No means to enforce separation of proof tree branches (cf. Coq bullets).
  - So case H3. might still work, while H3 now denotes sth. different.
  - Thus hard to track down where changes are required.
- Currently only a single specification may be imported into $G$.
- Extending $G$ with actual functions would also be nice.
- Why does Abella admit (with a warning) potentially consistency breaking inductive predicates with negative occurrences?
- Merging the Abella Proof General fork back into trunk would be desirably to avoid duplicate environments.
Summary

- **Contributions:**
  - Reduction of type formation and typing problems, formalised in Abella.
  - Comparison of de Bruijn and HOAS techniques for this proof.
  - Comparison of syntax translation via functions vs. relations.
  - Small usability study of the Abella theorem prover.

- **Current & Future Work:**
  - Rework Coq proof using relations instead of functions.
  - Improve HOAS support in Coq, see [Capretta & Felty ’06].
Thank you for your attention.

http://www.ps.uni-saarland.de/extras/ttt17-sysf/

Note: Presentation of the de Bruijn proof © CPP: Tuesday, January 17, 2017 – 17:00