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

Jonas Kaiser Gert Smolka

Saarland University,
Saarbrücken, Germany
{jkaiser,smolka}@ps.uni-saarland.de

Abstract

We give a machine-checked proof of the equivalence of the usual, two-sorted presentation of System F and its single-sorted pure type system variant $\lambda 2$. This is established by reducing the typability problem of F to $\lambda 2$ and vice versa. The systems are formulated using higher-order abstract syntax and the proof is executed in the Abella proof system. We compare and contrast this proof to our earlier Coq formalisation based on de Bruijn indices and context morphism lemmas.

Keywords Pure Type Systems, System F, HOAS, Relational Reasoning

1. The Problem

There are different presentations of "System F" in the literature, and we often found that properties that have been proven for one presentation are assumed for another, without verifying that the presentations are in fact equivalent.

In this paper we consider two presentations of System F we call F and $\lambda 2$. F is two-sorted and models Harper's [5] presentation of System F. $\lambda 2$ is a pure type system (PTS) and appears as a corner of Barendregt's λ-cube [2]. Our notion of equivalence is the *reduction of the typing problem* from one system to the other. As it turns out, proving this is surprisingly intricate.

2. Solution with de Bruijn in Coq

We recently [6] developed a machine-checked proof of this equivalence in Coq. In this prior work, we use de Bruijn syntax to handle the various variable binding mechanisms of the two systems. All essential lemmas of the proof development are phrased as context morphism lemmas (CML). We demonstrate that CMLs are not only a useful proof device to reason about structural properties of a single type system, but that they also scale to relating judgements of different type systems. We exploit this to precisely and systematically align F and $\lambda 2$. This Coq development, together with a preprint version of the associated paper, can be found at `https://www.ps.uni-saarland.de/extras/cpp17-sysf/`.

Our reduction theorem has the following form:

$$\vdash_{\mathsf{F}} s : A \iff \vdash_{2} \lfloor s \rfloor : \lfloor A \rfloor$$
$$\vdash_{2} a : b \iff \vdash_{\mathsf{F}} \lceil a \rceil : \lceil b \rceil$$

where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are suitable syntactic translation functions. These are necessary since the two systems use different syntactic languages.

The main difficulty of the proof is the correct handling of binders, which involves contexts, substitutions and their interaction with the translations. Doing this with a de Bruijn representation is technically involved but works well in Coq. The use of the Autosubst library [9] providing support for de Bruijn syntax turned out to be essential.

3. Solution with HOAS in Abella

An alternative to de Bruijn syntax is higher-order abstract syntax (HOAS). HOAS delegates the handling of binders to the underlying logic and thus avoids the need to model contexts and substitutions explicitly. The key idea is to model binders with a restricted class of functions and to inspect binders by applying them to fresh nominal constants. It is unclear whether the HOAS approach can be fully realised in a system like Coq (examples of approximate solutions are [3, 4]). There are, however, specialised systems whose underlying logic provides restricted function spaces and nominals as required by HOAS, like Twelf [7], Beluga [8] and Abella [1].

We undertook the project to redo the proof with a HOAS representation using the Abella system. Our interest was to find out how much HOAS simplifies the proof and how well the engineering of such a proof is supported by Abella. Our finding is that the overall proof structure carries over to Abella and that the technicalities of a de Bruijn representation are in fact eliminated. However, there are also severe drawbacks: The recursive translation functions must be represented as relations (obfuscating their computational status) and an induction principle for terms with binders is not available. Moreover, Abella has poor support for proof construction, compared to Coq. As is, the Abella proof is roughly as long as the Coq proof (we do not count the Autosubst library providing de Bruijn syntax for Coq) but more tedious to construct and fragile as it comes to refactoring.

There is one point where the Abella development clearly wins: The specification of the two underlying type systems is simpler since binders and contexts are provided by the underlying logic.

4. The Proof

Figures 1 and 2 show our HOAS specifications of F and $\lambda 2$. Abella is designed around a two-level logic approach and these definitions sit at the lower *specification level*, which is verbatim λProlog. We then reason about these structures at the *meta level*, where the logic is $\mathcal{G}$, the intuitionistic predicative fragment of Church's simple type theory, extended with natural induction, (co)inductive predicates and nominal quantification ($\nabla x.s$).

In [6] we use four syntactic translation functions to map the types and terms of F and $\lambda 2$. Abella is a relational system, so instead of functions we define, again at the specification level, two relations, $A \sim a$ for the types and $s \approx b$ for the terms. We then formulate the relational version of our reduction theorem as

$$\{s :_F A\} \iff \exists ab.\ \{s \approx a\} \wedge \{A \sim b\} \wedge \{a :_2 b\}$$
$$\{a :_2 b\} \iff \exists sA.\ \{s \approx a\} \wedge \{A \sim b\} \wedge \{s :_F A\}$$

Here $\{J\}$ is a meta level proposition that holds exactly when J is a derivable predicate at the specification level. This makes the inductive structure of λProlog derivations accessible at the meta level.

$$\begin{array}{rl}
\mathsf{Ty}_F, \mathsf{Tm}_F & \mathbf{type} \\
_ \to _ & \mathsf{Ty}_F \to \mathsf{Ty}_F \to \mathsf{Ty}_F \\
\forall . _ & (\mathsf{Ty}_F \to \mathsf{Ty}_F) \to \mathsf{Ty}_F \\
_ \,@\, _ & \mathsf{Tm}_F \to \mathsf{Tm}_F \to \mathsf{Tm}_F \\
\lambda _ . _ & \mathsf{Ty}_F \to (\mathsf{Tm}_F \to \mathsf{Tm}_F) \to \mathsf{Tm}_F \\
_ \,\overline{@}\, _ & \mathsf{Tm}_F \to \mathsf{Ty}_F \to \mathsf{Tm}_F \\
\Lambda . _ & (\mathsf{Ty}_F \to \mathsf{Tm}_F) \to \mathsf{Tm}_F \\
_ \, \mathbf{ty} & \mathsf{Ty}_F \to o \\
_ :_F _ & \mathsf{Tm}_F \to \mathsf{Ty}_F \to o
\end{array}$$

$$\frac{A\,\mathbf{ty} \qquad B\,\mathbf{ty}}{(A \to B)\,\mathbf{ty}} \qquad \frac{\mathbf{\Pi} x.\ x\,\mathbf{ty} \Rightarrow (A\,x)\,\mathbf{ty}}{(\forall . A)\,\mathbf{ty}}$$

$$\frac{s :_F A \to B \qquad t :_F A}{s \,@\, t :_F B} \qquad \frac{\mathbf{\Pi} x.\ x\,\mathbf{ty} \Rightarrow s\,x :_F A\,x}{\Lambda . s :_F \forall . A}$$

$$\frac{s :_F \forall . B \qquad A\,\mathbf{ty}}{s \,\overline{@}\, A :_F B\,A} \qquad \frac{A\,\mathbf{ty} \qquad \mathbf{\Pi} x.\ x :_F A \Rightarrow s\,x :_F B}{\lambda A . s :_F A \to B}$$

Figure 1. HOAS specification of F in Abella.

$$\begin{array}{rl}
\mathsf{Tm}_2 & \mathbf{type} \\
\Box & \mathsf{Tm}_2 \\
* & \mathsf{Tm}_2 \\
\Pi _ . _ & \mathsf{Tm}_2 \to (\mathsf{Tm}_2 \to \mathsf{Tm}_2) \to \mathsf{Tm}_2 \\
_ \,@\, _ & \mathsf{Tm}_2 \to \mathsf{Tm}_2 \to \mathsf{Tm}_2 \\
\lambda _ . _ & \mathsf{Tm}_2 \to (\mathsf{Tm}_2 \to \mathsf{Tm}_2) \to \mathsf{Tm}_2 \\
\mathcal{U}\ _ & \mathsf{Tm}_2 \to o \\
_ :_2 _ & \mathsf{Tm}_2 \to \mathsf{Tm}_2 \to o
\end{array}$$

$$\frac{}{\mathcal{U}\,\Box} \qquad \frac{}{\mathcal{U}\,*} \qquad \frac{}{* :_2 \Box} \qquad \frac{a :_2 \Pi c.d \qquad b :_2 c}{a \,@\, b :_2 d\,b}$$

$$\frac{a :_2 u \qquad \mathcal{U}\,u \qquad \mathbf{\Pi} x.\ x :_2 a \Rightarrow b\,x :_2 *}{\Pi a.b :_2 *}$$

$$\frac{\mathbf{\Pi} x.\ x :_2 a \Rightarrow c\,x :_2 * \qquad \begin{array}{c} a :_2 u \qquad \mathcal{U}\,u \\ \mathbf{\Pi} x.\ x :_2 a \Rightarrow b\,x :_2 c\,x \end{array}}{\lambda a.b :_2 \Pi a.c}$$

Figure 2. HOAS specification of $\lambda 2$ in Abella.

The basic proof design is to first establish that $A \sim a$ and $s \approx b$ are both *injective* and *functional*. We then proceed to show conditional *left- and right totality*, each coupled with judgement *preservation*. For the left-totality of the type relation this means given $\{A\,\mathbf{ty}\}$ holds, there exists an a, such that $A \sim a$ and $\{a :_2 *\}$ hold. The conditional right-totality for the type relation and the corresponding results for the term relation are analogue.

Throughout the proof we mostly deal with the more general propositions of the form $\{L \vdash J\}$, where L is a list of predicates that explicitly represents the implicit specification level reasoning context. Since such context lists can a priori contain arbitrary predicates, it is often necessary to restrict them to sensible hypotheses. Take for example a typing derivation in $\lambda 2$. The L_2 in $\{L_2 \vdash a :_2 b\}$ should be of the form $(n_1 :_2 c_1), \ldots, (n_k :_2 c_k)$, where the n_i are nominals that play the role of variables. This is achieved with a meta level inductive predicate $C_2(L_2)$, together with a suitable inversion principle for context extraction. We of course also have a predicate C_F for F context lists.

Notably, we also need a similar predicate C_R for statements of the form $\{L_R \vdash s \approx b\}$, and here we observe that a sensible context only contains premises of the form $n_a \sim n_b$ or $n_c \approx n_d$. In other words, a suitable context for a relation-derivation is a relation on type and term variables. These variable relations play the same role in the HOAS proof context morphisms play in the de Bruijn proof. In order to prove our preservation results, we have to connect the various context lists with a predicate $C(L_F, L_R, L_2)$. This predicate has to satisfy certain crucial properties, and these connect C closely to the context morphism lemmas we use in [6].

We observe that one of the most challenging cases of the proof appears to be independent from the choice of syntax encoding: The splitting of the single application mechanism of $\lambda 2$ into its two corresponding cases in F requires a certain degree of ingenuity in both of our proofs, as the required information is not immediately apparent in the associated typing derivations.

As a concluding remark we would like to point out a striking similarity with respect to how the final equivalences are obtained. We first show each of the forward implications and then use the respective other implication to complete the equivalences. Since in Abella we align the systems with relations rather than pairs of functions, the intricate cancellation laws of the Coq proof are not needed to complete this final step.

The accompanying Abella formalisation can be obtained from `https://www.ps.uni-saarland.de/extras/ttt17-sysf/`.

References

[1] David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu, and Yuting Wang. Abella: A system for reasoning about relational specifications. *Journal of Formalized Reasoning*, 7(2):1–89, 2014.

[2] Henk Barendregt. Introduction to generalized type systems. *Journal of Functional Programming*, 1(2):125–154, 1991.

[3] Venanzio Capretta and Amy P. Felty. Combining de Bruijn indices and higher-order abstract syntax in coq. In *Types for Proofs and Programs: International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, pages 63–77. Springer, 2007.

[4] Adam Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *ACM Sigplan Notices*, volume 43, pages 143–156. ACM, 2008.

[5] Robert Harper. *Practical foundations for programming languages*. Cambridge University Press, 2013.

[6] Jonas Kaiser, Tobias Tebbi, and Gert Smolka. Equivalence of System F and $\lambda 2$ in Coq based on context morphism lemmas (to appear). In *Proceedings of CPP 2017*. ACM, 2017.

[7] Frank Pfenning and Carsten Schürmann. System description: Twelf - A meta-logical framework for deductive systems. In *Automated Deduction - CADE-16, 16th International Conference on Automated Deduction, Trento, Italy, July 7-10, 1999, Proceedings*, pages 202–206. Springer, 1999.

[8] Brigitte Pientka and Joshua Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, pages 15–21. Springer, 2010.

[9] Steven Schäfer, Tobias Tebbi, and Gert Smolka. Autosubst: Reasoning with de Bruijn terms and parallel substitutions. In *Interactive Theorem Proving, Proceedings*, volume 9236 of *Lecture Notes in Computer Science*, pages 359–374. Springer, 2015.