Explicit names without α-equivalence: Simple type soundness for a CEK semantics.

Kenneth Knowles Cormac Flanagan
University of California at Santa Cruz
kknowles/cormac@cs.ucsc.edu

Motivation Proofs of metatheoretical properties of type systems using current techniques are notorious for straightforward but tedious inductions involving many subcases, encouraging proof elision. Ironically, as type systems are used for more complex languages and analyses, their published proofs get smaller. Today, we can expect the published soundness proof of a type system to look something like this:

Proof. Straightforward induction over the derivation. □

Proof assistants have the potential to aid programming language researchers in writing, checking, and even typesetting correct proofs. But effectively mapping handwritten proofs to a formal system can be unpredictable. While very simple proofs may be discharged automatically by proof search, at a very moderate level of complexity mechanized proofs can become much harder than their informal counterparts.

The difficulty of mechanizing metatheory proofs for programming languages lies in the definition of capture-avoiding substitution, which is expressed in informal proofs by considering terms equivalent up to renaming of bound variables. The POPLmark [2] challenge specifically targets this problem, and illustrates a variety of approaches, but none has achieved widespread dominance.

Contribution This work contributes another solution to naming: that of managing substitutions explicitly in a CEK-style operational semantics [4]. Our development does not include capture-avoiding substitution, and hence does not require a notion of α-equivalence, nor a fresh name supply. Our current result is a very concise and simple proof in Coq of *progress* and *preservation* for the simply-typed lambda calculus (STLC). In particular, our proof is about 1500 whitespace-delimited tokens[1], much shorter than the nearest comparable proof. Moreover, we use *no* custom tactics and only two non-trivial lemmas. This suggests that the overhead of managing explicit substitutions is less than that of the supporting lemmas and tactics that ease proofs using other representations.

Other Approaches to Naming We briefly compare representative examples of type soundness proofs for STLC using other approaches to naming, considering complexity, length, aesthetics, and applicability.

Aydemir proved soundness for STLC using the well-known Debruijn representation for terms, where each α-equivalence class has a unique representative [1]. The proofs in Coq constitute about 2600 tokens (667 lines) using three simple tactics, and over a dozen lemmas about lists and Debruijn indices. In addition to being about 70% longer than our solution, the statements of lemmas such as weakening are polluted with adjustments of Debruijn indices.

Charguéraud provides a soundness proof for STLC using a locally nameless representation [3]. Bound variables are represented with Debruijn indices, while explicit names are used for free variables. There is again a unique representative for each α-equivalence class, but now lemmas such as weakening are stated and proved in a natural way. The cost is that substitution must be defined both for names and for bound indices. Charguéraud's proof is over 2600 tokens of Coq (487 lines), using over twenty tactics and tactic notations for aggresive automation, and a number of lemmas about fresh names and relationships between the two definitions of substitution.

The Twelf wiki [5] presents an extremely concise proof of soundness for STLC using higher-order abstract syntax (HOAS). HOAS is a beautiful use of the binding implementation of the host language to encode binding for the object language. Unfortunately, HOAS is not supported by Coq or Isabelle because their logics become unsound in the presence of general non-strictly positive inductive types, so we do not consider HOAS a comparable technique.

Ongoing Work We are currently investigating the applicability and utility of our explicit substitutions approach to type systems with polymorphism and dependent types, where substitution takes place during type checking.

References

[1] B. Aydemir. Formalization of the meta-theory of a simply-typed lambda calculus using de bruijn indices. `http://www.cis.upenn.edu/proj/plclub/mmm/poplmark/baydemir-stlc.tar.gz`.

[2] B. E. Aydemir, A. Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. Mechanized metatheory for the masses: The poplmark challenge. In *Theorem Proving in Higher Order Logics*, pages 50–65, 2005.

[3] A. Charguéraud. Working with coq on the poplmark challenge. `http://www.chargueraud.org/arthur/research/2006/poplmark/`.

[4] M. Felleisen. *The Calculi of Lambda-v-CS Conversion: A Syntactic Theory of Control And State in Imperative Higher-Order Programming Languages.* PhD thesis, Indiana University, 1987.

[5] Proving metatheorems. `http://twelf.plparty.org/wiki/Proving_metatheorems`.

[1] 328 lines, but lines are easily compressed or expanded in Coq developments according to style. Coq's notatation mechanism makes even token counts suspect, giving only a very approximate measure of "ease of proof"