Autosubst: Automation for de Bruijn Substitutions

Steven Schäfer, Tobias Tebbi, and Gert Smolka

Saarland University

Formalizing syntactic theories with variable binders is not easy. We present AUTOSUBST [1], a library for the Coq proof assistant to automate this process. Given an inductive definition of syntactic objects in de Bruijn [7] representation augmented with binding annotations, AUTOSUBST synthesizes the substitution operations and automatically proves the basic lemmas about substitutions.

Our core contribution is the automation tactic `autosubst`, which simplifies and solves equations involving terms and substitutions. This makes the use of substitution lemmas unnecessary. The tactic is based on current work on a decision procedure for the equational theory of an extension of the σ-calculus by Adadi et. al. [2].

As in the σ-calculus, we use parallel substitutions. In our setting, a substitution σ is a function from de Bruijn indices to terms. The application $s[\sigma]$ of a substitution σ to a term s replaces every free variable in s according to σ. The parallel substitution operations admit an elegant recursive definition using the primitives of the σ-calculus. For instance, consider the untyped lambda calculus

$$\begin{aligned} x[\sigma] &= \sigma\, x \\ (s\,t)[\sigma] &= s[\sigma]\, t[\sigma] \\ (\lambda s)[\sigma] &= \lambda(s[0 \cdot (\sigma \circ \uparrow)]) \end{aligned} \qquad \begin{aligned} (s \cdot \sigma)\, 0 &= s \\ (s \cdot \sigma)\,(x+1) &= \sigma\, x \\ (\sigma \circ \tau)\, x &= (\sigma\, x)[\tau] \\ \uparrow x &= x + 1 \end{aligned}$$

where 0 is the first de Bruijn index and bound by the enclosing lambda, $\circ$ is the composition of substitutions, $\uparrow$ is the substitution that increases all de Bruijn indices by one, and $\cdot$ is the cons operation. However, this definition is not structurally recursive. Following Adams [3], we first restrict the definition to renamings and then obtain the full substitution application by replacing the problematic recursive call with a renaming. We hide the renaming operation that can occur after unfolding by eagerly replacing it with the ordinary substitution application.

As an example for the usage of `autosubst`, consider the following proof of the substitutivity of β-reduction in the untyped lambda calculus.

```
Lemma substitutivity s s’ : s ▷ s’ → ∀σ, s.[σ] ▷ s’.[σ].
Proof. induction 1; constructor; subst; autosubst. Qed.
```

The interesting subgoal solved by `autosubst` is the following parallel variant of Barendregt's substitution lemma [6, Lemma 2.1.16]

$$s[t \cdot \mathsf{id}][\sigma] = s[0 \cdot (\sigma \circ \uparrow)][t[\sigma] \cdot \mathsf{id}]$$

where id is the identity substitution. Note that $(\lambda s)\, t$ reduces to $s[t \cdot \mathsf{id}]$. The same proof script also works for the substitutivity of System F, both for substitutions that replace types and terms.

Our usage of parallel substitutions and a decision procedure are the main differences to related work [10, 9, 8, 5, 11]. To evaluate the impact of these design decisions, we have conducted the following case studies [1]:

- Progress and type preservation of System F with subtyping. This is the first part of the POPLmark challenge [4]. Since System F terms contain types and binders for type variables, we need to substitute terms in terms, types in types and types in terms. AUTOSUBST supports this, including support in `autosubst` for the interaction between the different substitution operations. The whole development fits into less than 500 line of code. This is about one third of the number of lines used by the solution of Vouillon [12], which also uses de Bruijn indices in Coq.
- Normalization of call-by-value System F. The proof uses logical relations, which always require some form of parallel substitutions in the fundamental theorem. The whole development fits into less than 200 lines.
- Type preservation of a Martin-Löf type theory. This is a technically challenging proof, which needs a number of related results (e.g. confluence of reduction).

In all of these case studies, the tactic `autosubst` takes care of all the tedium involving binders and substitutions. Using parallel substitutions, we obtain simple lemma statements without arithmetic or shifts.

AUTOSUBST is completely written in Coq. We synthesize the substitution operations using the meta-programming capabilities of Ltac. That is, we build a recursive procedure using the tactics `fix` and `destruct`. To do so, we need to inspect the annotations in the inductive definition of the syntax. These are realized as type-level identity functions. Since the types of the constructors contain the annotations, it suffices to determine the type of the constructor corresponding to the current subgoal of `destruct`. This is possible with the tactic `destruct s eqn:E`.

As future work, we want to add support for binders with variable arity (e.g., for pattern matching) and mutually inductive definitions of syntactic objects.

References

[1] Autosubst. `http://www.ps.uni-saarland.de/~ttebbi/autosubst`.

[2] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[3] R. Adams. Formalized metatheory with terms represented by an indexed family of types. In *Types for Proofs and Programs*, volume 3839 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2006.

[4] 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*, volume 3603 of *Lecture Notes in Computer Science*, pages 50–65. Springer Berlin Heidelberg, 2005.

[5] B. E. Aydemir and S. Weirich. LNgen: Tool support for locally nameless representations. Technical report, University of Pennsylvania, 2010.

[6] H. P. Barendregt. *The lambda calculus*, volume 3. North-Holland Amsterdam, 1984.

[7] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381 – 392, 1972.

[8] G. Lee, B. C. Oliveira, S. Cho, and K. Yi. Gmeta: A generic formal metatheory framework for first-order representations. In *Programming Languages and Systems*, volume 7211 of *Lecture Notes in Computer Science*, pages 436–455. Springer Berlin Heidelberg, 2012.

[9] F. Poittier. DBLIB, a Coq library for dealing with binding using de Bruijn indices. `http://gallium.inria.fr/~fpottier/dblib/dblib.tar.gz`, Dec. 2013.

[10] E. Polonowski. Automatically generated infrastructure for de Bruijn syntaxes. In *Interactive Theorem Proving*, volume 7998 of *Lecture Notes in Computer Science*, pages 402–417. Springer Berlin Heidelberg, 2013.

[11] P. Sewell, F. Z. Nardelli, S. Owens, G. Peskine, T. Ridge, S. Sarkar, and R. Strniša. Ott: Effective tool support for the working semanticist. *Journal of Functional Programming*, 20(1):71, 2010.

[12] J. Vouillon. A solution to the POPLmark challenge based on de Bruijn indices. *Journal of Automated Reasoning*, 49(3):327–362, 2012.