Advanced Science and Technology Letters Vol.41
(Mechanical Engineering 2013), pp.39-42
http://dx.doi.org/10.14257/astl.2013.41.10

Two Recent Approaches to Mechanical Developments of Formal Metatheory

Gyesik Lee

Hankyong National University,
Dept. of Computer & Web Information Engineering
Anseong-si, Gyonggi-do, 456-749 Korea
gslee@hknu.ac.kr

Abstract. This paper introduces two recent approaches to mechanical developments of formal metatheory for programming languages. The key issue concerns the representation and manipulation of terms with variable binding. This paper addresses first-order approaches only and gives an short overview of recent progresses in this field.

Keywords: Mechanization, variable binding, first-order representations, POPLMark challenge.

1 Introduction

A key issue in mechanical developments of formal metatheory for programming languages concerns the representation and manipulation of terms with variable binding. There are first-order and higher-order approaches.

The first-order approaches are close to pen-and-paper developments. However, they require tedious infrastructures for handling variable binding. For instance, in the locally nameless style [3] and locally named style [11], operations like substitution for parameters (free variables) and for (bound) variables as well some associated lemmas are necessary. And for de Bruijn indices [5] we need similar infrastructure.

Approaches based on higher-order abstract syntax (HOAS) [13, 7] are used in logical frameworks such as Abella [6], Hybrid [12] or Twelf [14]. In HOAS, the object-language binding is represented using the binding of the meta-language. This has the important advantage that facts about substitution or alpha-equivalence come for free since the binding infrastructure of the meta-language is reused.

In this paper we introduce two recent first-order approaches achieved by the author.

2 GMeta: a generic formal metatheory framework

The main drawback of first-order approaches is that the tedious infrastructure required for handling variable binding has to be repeated each time for a new object language. For each binding construct in the language, there is a set of *infrastructure operations* and associated *lemmas* that should be implemented.

ISSN: 2287-1233 ASTL

For example, System $F_{<:}$, which is the language described in the POPLMark challenge [2], requires 6 out of the 8 substitutions. Because for each operation we need to also prove a number of associated lemmas, solutions to the POPLMark challenge typically have a large percentage of lemmas and definitions just for infrastructure. In the solution by Aydemir et al. [3], infrastructure amounts to 65% of the total number of definitions and lemmas.

To deal with the combinatorial explosion of infrastructure operations and lemmas, Lee et al. [8] proposed a generic framework called GMeta for first-order representations of variable binding. GMeta makes use of *datatype-generic programming* (DGP) and *modular programming* techniques. The key idea is that, with DGP, one can define once and for all the tedious infrastructure lemmas and operations in a generic way and, with modules, one can provide a convenient interface for users to instantiate such generic infrastructure to their object languages.

GMeta is implemented in the proof assistant Coq. In GMeta, a DGP *universe* [10] is used to represent a large family of object languages and includes constructs for representing the binding structure of those languages. The universe is independent of the particular choice of first-order representations: it can be instantiated, for example, to *locally nameless* or *de Bruijn* representations. GMeta uses that universe to provide libraries with the infrastructure for various first-order representations.

The infrastructure is reused by users through so-called *templates*. Templates are functors parameterized by isomorphisms between the object language and the corresponding representation of that language in the universe. By instantiating templates with isomorphisms, users get access to a module that provides infrastructure tailored for a particular binding construct in their own object language.

		style	savings
STLC	GMeta vs Aydemir et al.	LN	52%
$F_{<:}$	GMeta vs Aydemir et al.	LN	38%
	GMeta vs Vouillon	dB	35%

Fig. 1. Savings in terms of numbers of definitions and lemmas.

In order to verify the effectiveness of GMeta in reducing the infrastructure overhead, GMeta is used for case studies using locally nameless and de Bruijn representations. The two case studies are a solution to the POPLMark challenge parts 1A+2A, and a formalization of the STLC.

GMeta can reduce the infrastructure overhead because it provides reuse of boilerplate definitions and lemmas. By boilerplate we mean *common operations*, *lemmas about common operations*}, and *lemmas involving well-formedness*.

The biggest benefit of GMeta is that it significantly lowers the overheads required in mechanical formalizations by providing reuse of the basic infrastructure. Figure 1 shows the savings that GMeta achieved relative to the reference solutions by Aydemir et al. [3] and Vouillon [18]. Note that in GMeta only user-defined code is counted. In all case studies more than 35% of the total numbers of definitions were saved. We conducted case studies in both System $F_{<:}$ and STLC.

3 Stoughton-style nominal approach

Stoughton [16] demonstrated a non-canonical, but elegant and sound nominal approach to variable binding. Any substitution (including identity substitution) puts terms in a substitution normal form, and alpha-congruence is equality of substitution normal forms.

In spite of the non-canonical aspect, Stoughton's representation is very interesting because it is closely related to Barendregt's Variable Convention:

VARIABLE CONVENTION. If M1, ..., Mn occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables. (Barendregt [4])

This convention tacitly assumes that bound variables can be renamed by fresh ones when necessary, for example, in case of possible variable capture. On the other hand, in Stoughton's representation bound variables are to be renamed whenever we do substitutions without thinking of whether variable capture occurs or not.

From the standpoint of mechanization, it is much more convenient to do reasoning without being based on tacit conventions. Indeed, as long as we stick to one-sorted variable names, there is no direct way to formalize reasoning based on such tacit assumptions like Barendregt's Variable Convention. Such a formalization would work entirely at the level of terms quotiented by alpha-congruence. Pitts's Nominal Logic [15] and nominal techniques in Isabelle/HOL [17], e.g., can be regarded as methods of formalizing the spirit of Barendregt.

Lee and Na [9] used [16] as a test case in exploring a nominal approach to variable binding in Coq. This approach is characterized by a close correspondence between the original reasoning on paper and ours within Coq. They showed that it is not only feasible, but also convenient to work with the nominal approach in Coq. Most of all, our formalization is very compact and close to the original reasoning on paper.

The formalization in Coq was done as close as possible to the original reasoning on paper and showed that it is not only feasible, but also convenient to work with the nominal approach in Coq. In particular, Nominal approach with one-sorted variable names is applied to variable binding in Coq. Coq's internal equality is used instead of any kind of set-theoretic equalities or setoid equalities. Only Coq's basic terminology and techniques are used.

4 Conclusion

We introduced two recent approaches in mechanical developments of formal metatheory for programming languages: GMeta and Stoughton-style nominal approach. GMeta improves on the development of the infrastructure for a new formalization, whereas Stoughton-style nominal approach makes it plausible to use one sort of variables without much worrying about variable capture. It remains to extend these works by applying the two approaches to more realistic examples.

References

1. Aydemir, B., Weirich, S., Zdancewic, S.: Abstracting syntax. Technical Report MS-CIS-09-06, University of Pennsylvania (2009)
2. Aydemir, B. E., Bohannon, A., Fairbairn, M., Nathan Foster, N., Pierce, B. C., Sewell, P., Vytiniotis, D., Washburn, G., Weirich, S., Zdancewic, S.; Mechanized Metatheory for the Masses: The POPLmark Challenge. In: TPHOLs '05 (2005)
3. Aydemir, B. E., Charguéraud, A., Pierce, B. C., Pollack, R., Weirich, S.: Engineering formal metatheory. In: POPL '08 (2008)
4. Barendregt, H. P.: The Lambda Calculus - Its Syntax and Semantics. North-Holland (1984)
5. De Bruijn, N. G.: 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)
6. Gacek, A.: The Abella Interactive Theorem Prover (System Description). In: IJCAR '08 (2008)
7. Harper, R., Honsell, F., Plotkin, G.: A framework for defining logics. J. ACM, 40(1):143184 (1993)
8. Lee, G., Oliveira, B. C. d. S., Cho, S., Yi, K.: Gmeta: A generic formal metatheory framework for first-order representations. In: ESOP, pages 436-455 (2012)
9. Lee G., Na, H.: Using nominal reasoning techniques in the coq proof assistant. Journal of KIISE: Software and Applications, 40(11):751-756 (2013)
10. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis (1984)
11. McKinna, J., Pollack, R.: Pure type systems formalized. In: TLCA '93, pages 289-305. Springer-Verlag (1993)
12. Momigliano, A., Martin, A. J., Felty, A. P.: Two-level hybrid: A system for reasoning using higher-order abstract syntax. Electron. Notes Theor. Comput. Sci., 196:85-93 (2008)
13. Pfenning, F., Elliot, C.: Higher-order abstract syntax. In: PLDI '88 (1988)
14. Pfenning, F., Schürmann, C.: System description: Twelf - a meta-logical framework for deductive systems. In: CADE '99 (1999)
15. Pitts, A. M.: Nominal logic, a first order theory of names and binding. Inf. Comput., 186(2):165-193 (2003)
16. Stoughton, A.: Substitution Revisited. Theor. Comput. Sci., 59:317-325 (1988)
17. Urban, C., Tasson, C.: Nominal Techniques in Isabelle/HOL. In: CADE '05 (2005)
18. Vouillon, J.: Poplmark solutions using de Bruijn indices (2007) Available at https://alliance.seas.upenn.edu/~plclub/cgi-bin/poplmark/.

