It Is Time to Mechanize Programming Language Metatheory*

Benjamin C. Pierce[1], Peter Sewell[2], Stephanie Weirich[1], and Steve Zdancewic[1]

[1] Department of Computer and Information Science, University of Pennsylvania
[2] Computer Laboratory, University of Cambridge

Abstract. How close are we to a world in which mechanically verified software is commonplace? A world in which theorem proving technology is used routinely by both software developers and programming language researchers alike? One crucial step towards achieving these goals is mechanized reasoning about language metatheory. The time has come to bring together the theorem proving and programming language communities to address this problem. We have proposed the POPLMark challenge as a concrete set of benchmarks intended both for measuring progress in this area and for stimulating discussion and collaboration. Our goal is to push the boundaries of existing technology to the point where we can achieve mechanized metatheory for the masses.

1 Mechanized Metatheory for the Masses

One significant obstacle to achieving the goal of verified software is reasoning about the languages in which the software is written. Without formal models of programming languages, it is impossible to even state, let alone prove, meaningful properties of software or tools such as compilers. It is therefore essential that we develop appropriate tools for modeling programming languages and mechanically checking their metatheoretic properties. This infrastructure should provide facilities for proving properties of operational semantics, program analyses (such as type checkers), and program transformations (such as optimization and compilation).

Many proofs about programming languages are straightforward, long, and tedious, with just a few interesting cases. Their complexity arises from the management of many details rather than from deep conceptual difficulties; yet small mistakes or overlooked cases can invalidate large amounts of work. These effects are amplified as languages scale: it becomes very hard to keep definitions and proofs consistent, to reuse work, and to ensure tight relationships between theory and implementations. Automated proof assistants offer the hope of significantly easing these problems. However, despite much encouraging progress in recent years and the availability of several mature tools (ACL2 [15], Coq [2], HOL [10], Isabelle [20], Lego [16], NuPRL [5], PVS [21], Twelf [22], etc.), their use is still not commonplace.

* This position paper is adapted from the introduction to the POPLMark Challenge paper [1].

B. Meyer and J. Woodcock (Eds.): Verified Software, LNCS 4171, pp. 26–30, 2008.

We believe that the time is right to join the efforts of the two communities, bringing developers of automated proof assistants together with a large pool of eager potential clients—programming language designers and researchers. In particular, we intend to answer two questions:

1. What is the current state of the art in formalizing language metatheory and semantics? What can be recommended as best practices for groups (typically not proof-assistant experts) embarking on formalized language definitions, either small- or large-scale?
2. What improvements are needed to make the use of tool support commonplace? What can each community contribute?

Over the past six months, we have attempted to survey the landscape of proof assistants, language representation strategies, and related tools. Collectively, we have applied automated theorem proving technology to a number of problems, including proving transitivity of the algorithmic subtype relation in Kernel $F_{\leq}$ [4,3,6], proving type soundness of Featherweight Java [14], proving type soundness of variants of the simply typed λ-calculus and $F_{\leq}$, and a substantial formalization of the behavior of TCP, UDP, and the Sockets API. We have carried out these case studies using a variety of object-language representation strategies, proof techniques, and proving environments. We have also experimented with lightweight tools designed to make it easier to define and typeset both formal and informal mathematics. Although experts in programming language theory, we were (and are) relative novices with respect to computer-aided proof.

Our conclusion from these experiments is that the relevant technology has developed *almost* to the point where it can be widely used by language researchers. We seek to push it over the threshold, making the use of proof tools common practice in programming language research—mechanized metatheory for the masses.

Tool support for formal reasoning about programming languages would be useful at many levels:

1. *Machine-checked metatheory.* These are the classic problems: type preservation and soundness theorems, unique decomposition properties for operational semantics, proofs of equivalence between algorithmic and declarative variants of type systems, etc. At present such results are typically proved by hand for small to medium-size calculi, and are not proved at all for full language definitions. We envision a future in which the papers in conferences such as *Principles of Programming Languages (POPL)* and the *International Conference on Functional Programming (ICFP)* are routinely accompanied by mechanically checkable proofs of the theorems they claim.
2. *Use of definitions as oracles for testing and animation.* When developing a language implementation together with a formal definition one would like to use the definition as an oracle for testing. This requires tools that can decide typing and evaluation relationships, and they might differ from the tools used for (1) or be embedded in the same proof assistant. In some cases one could use a definition directly as a prototype.

3. *Support for engineering large-scale definitions.* As we move to full language definitions—on the scale of Standard ML [17] or larger—pragmatic "software engineering" issues become increasingly important, as do the potential benefits of tool support. For large definitions, the need for elegant and concise notation becomes pressing, as witnessed by the care taken by present-day researchers using informal mathematics. Even lightweight tool support, without full mechanized proof, could be very useful in this domain, e.g. for sort checking and typesetting of definitions and of informal proofs, automatically instantiating definitions, performing substitutions, etc.

Our goal is to stimulate progress in this area by providing a common framework for comparing alternative technologies. Our approach has been to design a set of challenge problems, dubbed the POPLMark Challenge [1], chosen to exercise many aspects of programming languages that are known to be difficult to formalize: variable binding at both term and type levels, syntactic forms with variable numbers of components (including binders), and proofs demanding complex induction principles. Such challenge problems have been used in the past within the theorem proving community to focus attention on specific areas and to evaluate the relative merits of different tools; these have ranged in scale from benchmark suites and small problems [23, 11, 7, 13, 9, 19] up to the grand challenges of Floyd, Hoare, and Moore [8, 12, 18]. We hope that our challenge will have a similarly stimulating effect.

The POPLMark problems are drawn from the basic metatheory of a call-by-value variant of System $F_{\leq}$ [3, 6], enriched with records, record subtyping, and record patterns. Our challenge provides an informal-mathematics definition of its type system and operational semantics and outline proofs of some of its metatheory. This language is of moderate scale—neither a toy calculus nor a full-blown programming language—to keep the work involved in attempting the challenges manageable.[1] The intent of this challenge is to cover a broad range of issues that arise in the *formalization* of programming languages; of course there are many programming language *features*, such as control-flow operators, state, and concurrency, not covered by our sample problem, but we believe that a system capable of formalizing the POPLMark problems should be able to formalize those features as well. Nevertheless, we expect this challenge set to grow and evolve as the community addresses some problems and discovers others.

The initial POPLMark challenge has already been disseminated to a wide audience of theorem prover and programming language researchers. We are in the process of collecting and evaluating solutions. Those results, along with related information about mechanized metatheory, will be available on our web site.[2] In the longer run, we hope that this site, and the corresponding mailing list [3] will serve as a forum for promoting and advancing the current best practices in

[1] Our challenges therefore explicitly address only points (1) and (2) above; we regard the pragmatic issues of (3) as equally critical, but it is not yet clear to us how to formulate a useful challenge problem at this larger scale.

[2] `http://www.cis.upenn.edu/proj/plclub/mmm/`

[3] `poplmark@lists.seas.upenn.edu`

proof assistant technology and making this technology available to the broader programming languages community and beyond. We encourage researchers to try out the POPLMark Challenge using their favorite tools and send us their solutions for inclusion in the web site.

References

1. Aydemir, B.E., Bohannon, A., Fairbairn, M., Foster, J.N., Pierce, B.C., Sewell, P., Vytiniotis, D., Washburn, G., Weirich, S., Zdancewic, S.: Mechanized metatheory for the masses: The POPLmark challenge. In: Theorem Proving in Higher Order Logics, 18th International Conference, Oxford, UK (August 2005)
2. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. In: EATCS Texts in Theoretical Computer Science, vol. XXV, Springer, Heidelberg (2004)
3. Cardelli, L., Martini, S., Mitchell, J.C., Scedrov, A.: An extension of System F with subtyping. In: Ito, T., Meyer, A.R. (eds.) TACS 1991. LNCS, vol. 526, pp. 750–770. Springer, Heidelberg (1991)
4. Cardelli, L., Wegner, P.: On understanding types, data abstraction, and polymorphism. Computing Surveys 17(4), 471–522 (1985)
5. Constable, R.L., Allen, S.F., Bromley, M., Cleaveland, R., Cremer, J.F., Harper, R.W., Howe, D.J., Knoblock, T.B., Mendler, P., Panangaden, P., Sasaki, J.T., Smith, S.F.: Implementing Mathematics with the NuPRL Proof Development System. Prentice-Hall, Englewood Cliffs, NJ (1986)
6. Curien, P.-L., Ghelli, G.: Coherence of subsumption: Minimum typing and typechecking in F. Mathematical Structures in Computer Science. In: Gunter, C.A., Mitchell, J.C. (eds.) Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design, vol. 2, pp. 55–91. MIT Press, Cambridge (1994)
7. Dennis, L.A.: Inductive challenge problems (2000), `http://www.cs.nott.ac.uk/~lad/research/challenges`
8. Floyd, R.W.: Assigning meanings to programs. In: Schwartz, J.T. (ed.) Mathematical Aspects of Computer Science. Proceedings of Symposia in Applied Mathematics, vol. 19, pp. 19–32. American Mathematical Society, Providence, Rhode Island (1967)
9. Gent, I.P., Walsh, T.: CSPLib: a benchmark library for constraints. Technical report, Technical report APES-09-, 1999. A shorter version appears in the Proceedings of the 5th International Conference on Principles and Practices of Constraint Programming (CP-99) (1999), `http://csplib.cs.strath.ac.uk/`
10. Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: a theorem proving environment for higher order logic. Cambridge University Press, Cambridge (1993)
11. Green, I.: The dream corpus of inductive conjectures (1999), `http://dream.dai.ed.ac.uk/dc/lib.html`
12. Hoare, T.: The verifying compiler: A grand challenge for computing research. J. ACM 50(1), 63–69 (2003)
13. Hoos, H., Stuetzle, T.: Satlib, `http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/`
14. Igarashi, A., Pierce, B., Wadler, P.: Featherweight Java: A minimal core calculus for Java and GJ. In: ACM SIGPLAN Conference on Object Oriented Programming: Systems, Languages, and Applications (OOPSLA) October 1999. Full version in ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 23(3) (May 2001)

15. Kaufmann, M., Moore, J.S., Manolios, P.: Computer-Aided Reasoning: An Approach. Kluwer Academic Publishers, Dordrecht (2000)
16. Luo, Z., Pollack, R.: The LEGO proof development system: A user's manual. Technical Report ECS-LFCS-92-211, University of Edinburgh (May 1992)
17. Milner, R., Tofte, M., Harper, R., MacQueen, D.: The Definition of Standard ML, Revised edition. MIT Press, Cambridge (1997)
18. Moore, J.S.: A grand challenge proposal for formal methods: A verified stack. In: Aichernig, B.K., Maibaum, T.S.E. (eds.) Formal Methods at the Crossroads. From Panacea to Foundational Support. LNCS, vol. 2757, pp. 161–172. Springer, Heidelberg (2003)
19. Moore, J.S., Porter, G.: The apprentice challenge. ACM Trans. Program. Lang. Syst. 24(3), 193–216 (2002)
20. Nipkow, T., Paulson, L.C., Wenzel, M.T.: Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002)
21. Owre, S., Rajan, S., Rushby, J.M., Shankar, N., Srivas, M.K.: PVS: Combining specification, proof checking, and model checking. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 411–414. Springer, Heidelberg (1996)
22. Pfenning, F., Schürmann, C.: System description: Twelf — A meta-logical framework for deductive systems. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 202–206. Springer, Heidelberg (1999)
23. Sutcliffe, G., Suttner, C.: The TPTP problem library. Journal of Automated Reasoning 21(2), 177–203 (1998)