HAL
open science

Compiler verification for fun and profit

Xavier Leroy

To cite this version:

Xavier Leroy. Compiler verification for fun and profit. FMCAD 2014 - Formal Methods in Computer-Aided Design, Oct 2014, Lausanne, Switzerland. pp.9. hal-01076547

HAL Id: hal-01076547

https://inria.hal.science/hal-01076547v1

Submitted on 22 Oct 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Compiler verification for fun and profit

Xavier Leroy
Inria Paris–Rocquencourt,
Domaine de Voluceau, BP 105, 78153 Le Chesnay, France
Email: xavier.leroy@inria.fr

Abstract of invited talk

Formal verification of software or hardware systems — be it by model checking, deductive verification, abstract interpretation, type checking, or any other kind of static analysis — is generally conducted over high-level programming or description languages, quite remote from the actual machine code and circuits that execute in the system. To bridge this particular gap, we all rely on *compilers* and other code generators to automatically produce the executable artifact. Compilers are, however, vulnerable to *miscompilation*: bugs in the compiler that cause incorrect code to be generated from a correct source code, possibly invalidating the guarantees so painfully obtained by source-level formal verification. Recent experimental studies [1] show that many widely-used production-quality compilers suffer from miscompilation.

The formal verification of compilers and related code generators is a radical, mathematically-grounded answer to the miscompilation issue. By applying formal verification (typically, interactive theorem proving) to the compiler itself, it is possible to guarantee that the compiler preserves the semantics of the source programs it transforms, or at least preserves the properties of interest that were formally verified over the source programs. Proving the correctness of compilers is an old idea [2], [3] that took a long time to scale all the way to realistic compilers. In the talk, I give an overview of CompCert C [4], a moderately-optimizing compiler for almost all of the ISO C 99 language that has been formally verified using the Coq proof assistant [5].

The CompCert project is one point in a space of code generators whose verification deserves attention. For example, functional languages and object-oriented languages raise the issue of jointly verifying the compiler and the run-time system (memory management, exception handling, etc) that the generated code depends on. At the other end of the expressiveness spectrum, synchronous languages and hardware description languages also raise interesting verified generation issues, as exemplified by Pnueli's seminal work on translation validation for Signal [6] and Braibant and Chlipala's recent work on verified hardware synthesis [7].

Orthogonally, the integration of verification tools and compilers that are both verified against a shared formal semantics opens fascinating opportunities for "super-optimizations" that generate better code by exploiting the properties of the source code that were formally verified.

References

[1] X. Yang, Y. Chen, E. Eide, and J. Regehr, "Finding and understanding bugs in C compilers," in *PLDI 2011: Programming Language Design and Implementation*. ACM, 2011, pp. 283–294.

[2] J. McCarthy and J. Painter, "Correctness of a compiler for arithmetical expressions," in *Mathematical Aspects of Computer Science*, ser. Proc. of Symposia in Applied Mathematics, vol. 19. American Mathematical Society, 1967, pp. 33–41.

[3] R. Milner and R. Weyrauch, "Proving compiler correctness in a mechanized logic," in *Proc. 7th Annual Machine Intelligence Workshop*, ser. Machine Intelligence, B. Meltzer and D. Michie, Eds., vol. 7. Edinburgh University Press, 1972, pp. 51–72.

[4] X. Leroy, "Formal verification of a realistic compiler," *Communications of the ACM*, vol. 52, no. 7, pp. 107–115, 2009.

[5] Coq development team, "The Coq proof assistant," Software and documentation available at http://coq.inria.fr/, 1989–2014.

[6] A. Pnueli, O. Strichman, and M. Siegel, "Translation validation for synchronous languages," in *ICALP'98: Automata, Languages and Programming*, ser. LNCS, vol. 1443. Springer, 1998, pp. 235–246.

[7] T. Braibant and A. Chlipala, "Formal verification of hardware synthesis," in *CAV 2013: Computer Aided Verification*, ser. LNCS, vol. 8044. Springer, 2013, pp. 213–228.

Acknowledgments

This work was supported by the VERASCO project (ANR-11-INSE-003) of Agence Nationale de la Recherche.