CFG Patterns: A new tool to formally verify optimisations in Vellvm

Yannick ZAKOWSKI, Inria CASH/LIP
Gabriel RADANNE, Inria CASH/LIP

1 Context

Fifteen years ago, CompCert [1] demonstrated that the decades-old dream [2] of a verified compiler was within grasp. Over the years, Leroy et al. indeed built an end-to-end optimizing C99 compiler coded and proved correct in the Coq proof assistant [4]. The proof of correction establishes that the assembly program resulting from the compilation only exhibits dynamic behaviors already present in the source program: the compiler preserves functional correctness.

This seminal work spawned in its wake a vast body of works on verified compilation. Vellvm is an ambitious project in this design space. It aims in the long run to formalize and verify part of the LLVM compilation infrastructure, and focus in particular its early efforts on the formalization of LLVM IR, the intermediate language at the core of LLVM. Beyond the specific language of interest, the modern iteration of the Vellvm project [6] has the originality to explore the use of modern semantic techniques based on the Interaction Trees [5], a recently introduced generic Coq library to design denotational domains. With these modern semantic tools come novel proof techniques in the realm of verified compilers, putting an emphasis on algebraic approaches.

Despite these new techniques, concretely expressing the transformations and optimizations involved in a production-grade compiler is still very difficult. Expressing it requires to concretely walk through, deconstruct and rebuild the control flow graph of the program. Doing so amount to detect the patterns of interest and if applicable perform the adequate substitution – this difficulty is already encountered in traditional compilers. In addition and specific to verified compilers, this manual massaging of the CFG does not yield itself well to proofs, as the semantic invariants of the patterns detected are often obscured by the nitty-gritty bits of the transformations.

2 Objective of the Internship

Recent works have led to the development of *combinators* [3], allowing to build control flow graphs functionally. These combinators provide transparent name handling, simplifying both the definition of the compilation passes and the related proofs. The resulting DSL gives an elegant and convenient way to *build* control flow graphs. We have in CASH already conducted preliminary works to develop similar combinators, in Coq, over Vellvm's internal representation of CFGs. These combinators are equipped with semantic equations characterizing their behavior, easing the reasoning about programs built on them.

But symmetrically, an optimization needs to know what to do with a given arbitrary CFG : we would hence like to *deconstruct* a control flow graph using a *pattern-like language*.

The goal of this internship is precisely to design and implement such a pattern language. A pattern would match a piece of the control flow graph, extract some of its sub-parts, and carry the necessary information to recombine these components – typically through combinators in the style mentioned above – as well as show the semantic relationship between all these components.

The first expected concrete step will be to design a pattern language acting as a mirror of the functional combinators in order to build programs as a control flow graph – new combinators can be introduced as needed on the way. A second step will be to code up an efficient way to find a given pattern in an arbitrary graph, and prove the correction of this function by expressing the relationship between the graph and the computed deconstructed form. Finally, we will evaluate our design on simple program transformations.

LIP – UMR CNRS / ENS Lyon / UCB Lyon 1 / Inria – 69007 Lyon
E-mail : yannick.zakowski@inria.fr – gabriel.radanne@inria.fr

3 Profile

The candidate should be familiar with formal approach to programming languages, and have a taste for both compilation and formal proofs.

On the practical side, an experience in software development is welcome, along with a minimal knowledge of collaborative tools such as `git`. Knowledge of Coq is also required.

4 Contact and Localization

The internship will take place at ENS Lyon, in the LIP laboratory. Contact Yannick ZAKOWSKI (yannick.zakowski@inria.fr) and Gabriel RADANNE (gabriel.radanne@inria.fr) if you are interested in this internship.

References

[1] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009. doi: 10.1145/1538788.1538814. URL `http://doi.acm.org/10.1145/1538788.1538814`.

[2] John McCarthy and James Painter. Correctness of a compiler for arithmetic expressions. *Mathematical aspects of computer science*, 1, 1967.

[3] Arjen Rouvoet, Robbert Krebbers, and Eelco Visser. Intrinsically typed compilation with nameless labels. *Proc. ACM Program. Lang.*, 5(POPL):1–28, 2021. doi: 10.1145/3434303. URL `https://doi.org/10.1145/3434303`.

[4] The Coq Development Team. The coq proof assistant, version 8.11.0, January 2020. URL `https://doi.org/10.5281/zenodo.3744225`.

[5] Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C Pierce, and Steve Zdancewic. Interaction trees: representing recursive and impure programs in Coq. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–32, 2019.

[6] Yannick Zakowski, Calvin Beck, Irene Yoon, Vadim Zaliva, Ilia Zaichuk, and Steve Zdancewic. Modular, compositional, and executable formal semantics for llvm ir. *In Submission*.