Static analyses for Vellvm

Laure Gonnord, Université Lyon1/CASH/ LIP
Yannick Zakowski Inria CASH/LIP

2020

Keywords

Abstract interpretation, compilation, abstract domain, certified compilation, LLVM, Vellvm

General Context

Optimizing compilers have been argued to be among the most complex humanly-built objects. In all account, they have been consistently observed to exhibit difficult to diagnose bugs [7], despite being thoroughly tested. CompCert [3] in contrast has paved the road to fully verified optimizing compilers : the compiler is coded and proved correct in a proof assistant, Coq for instance.

Vellvm [9] has been initiated in 2013 as a project similar in fashion to CompCert, but aiming to formalize parts of the LLVM [2] infrastructure. Central to this project is the definition of a formal semantics of (the sequential fragment of) LLVM IR. More recently, Zakowski et. al [8] [1] have completely rebuilt the semantics of Vellvm based on modern semantics tools, the *Interaction Trees* [6] more specifically. Rather than relying on a traditional small-step operational semantics, a denotational semantics is built in a modular fashion.

The resulting semantics admits appealing compositional reasoning principles, as well as a very expressive relational program logics allowing for the construction of arbitrary refinements, allowing for the proof of semantic preservation of transformations.

While this new semantics has been so far exercised through the ongoing proof of a front-end for the Helix project [2] and for elementary optimizations over IR, no static analyses has been developed yet to infer invariants of llvm code.

We thus propose to explore the possibilities opened by the new semantics in terms of abstract-interpretation based static analyses. Depending on the intern, the project may involve major Coq development or more traditional “on paper” proofs. However we aim to have a final working proof of concept.

Location and supervision

The internship will take place in LIP, Lyon. It will be co-advised by Yannick Zakowski and Laure Gonnord, in the CASH team.

The two supervisors have experience in developing SSA-based static analyses (Laure Gonnord) and developing formal proofs of software in Coq (Yannick Zakowski).

Internship content

Trough this internship, we propose three possible exploratory projects :

— to develop unverified static analyses for LLVM programs aiming at scale and compositionality, in the spirit of sparse abstract interpretation for numerical or memory properties [5, 4]. In this idea, the connection to vellvm would be to use its formal semantics as an auxiliary tool to rationally design the analysis. Additionally, we encourage the analysis to be coded in OCaml to allow for plugin it with Vellvm, even if unverified.

1. https://github.com/vellvm/vellvm
2. https://github.com/vzaliva/helix/

LIP – UMR CNRS / ENS Lyon / UCB Lyon 1 / Inria – 69007 Lyon
E-mail : Laure.Gonnord@ens-lyon.fr et Yannick Zakowski@ens-lyon.fr

— to develop validated static analyses for Vellvm. The analysis would be coded in OCaml, and unverified itself. However, it would produce witnesses suitable to be checked. A checker, coded in Coq, takes such a witness as input and decides if the witness is valid or not. The checker is itself proved : if it validates the witness, the invariant is guaranteed to hold semantically. With this option, the first objective of the internship would be the design of the analysis, the notion of witness produced and the checker. The formal verification of the checker would be considered as a secondary objective.
— to develop verified static analyses for Vellvm. The analysis would therefore be itself coded in Coq, and proved correct in the style of Verasco [1].

The intership will consist of the following steps :
— Understand the software architecture and the main semantic ideas of Vellvm.
— Study sparse dataflow analysis and variants to understand and propose variants suited to the Vellvm semantics.
— Chose the direction according to your taste and the previous steps.

Références

[1] Jacques-Henri Jourdan, Vincent Laporte, Sandrine Blazy, Xavier Leroy, and David Pichardie. A formally-verified c static analyzer. *SIGPLAN Not.*, 50(1) :247–259, January 2015.

[2] Chris Lattner and Vikram Adve. LLVM : A compilation framework for lifelong program analysis and transformation. pages 75–88, San Jose, CA, USA, Mar 2004.

[3] Xavier Leroy, Andrew W. Appel, Sandrine Blazy, and Gordon Stewart. The CompCert Memory Model, Version 2. Research Report RR-7987, INRIA, June 2012.

[4] Maroua Maalej Kammoun. *Low-cost memory analyses for efficient compilers*. Theses, Université de Lyon, September 2017.

[5] Henrique Nazaré, Izabela Maffra, Willer Santos, Leonardo Oliveira, Fernando Magno Quintão Pereira, and Laure Gonnord. Validation of Memory Accesses Through Symbolic Analyses. In *ACM International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA'14)*, pages 791–809, Portland, Oregon, United States, October 2014.

[6] Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. Interaction trees. *Proceedings of the ACM on Programming Languages*, 4(POPL), 2020.

[7] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. Finding and understanding bugs in c compilers. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, page 283–294, New York, NY, USA, 2011. Association for Computing Machinery.

[8] Yannick Zakowski, Calvin Beck, Vadim Zaliva, Irene Yoon, Ilia Zaichuk, and Steve Zdancewic. Modular, Compositional, and Executable Fromal Semantics for LLVM IR. Draft RR-7987, INRIA, June 2012.

[9] Jianzhou Zhao, Santosh Nagarakatte, Milo M. K. Martin, and Steve Zdancewic. Formalizing the LLVM Intermediate Representation for Verified Program Transformations. In *Proc. of the ACM Symposium on Principles of Programming Languages (POPL)*, 2012.