A parameterized bisimulation for interaction trees

Yannick Zakowski Inria CASH/LIP and Ludovic Henrio CNRS CASH/LIP

2020

Keywords

Algebraic effects, itrees, bisimulation, Coq

General Context

Acknowledging that traditional optimizing compilers *are* buggy [4], *verified* optimizing compilers seek to be programmed and formally proved correct in a proof assistant. The approach fundamentally relies on the simple observation that a compiler is staged as a sequence of program transformations either optimizing the code, or translating it down to a lower level language. Each such transformation can be proved correct in isolation, and these theorems of correctness composed to establish the correctness of the compiler as a whole.

To prove a program transformation correct consists in proving that it preserves the semantics of the program: any behavior that the resulting program exhibits was already present in the original program. As such, vast classes of safety properties established at the source are guaranteed to be equally valid over the result of the (verified) compilation process. A cornerstone in verified compilation is therefore the definition of the formal semantics of the intermediate languages involved. Most existing approaches have followed the seminal work of CompCert [2]: an operational small step semantics specified propositionally is used. The approach have many benefits: it is easy to mechanize, well understood, and comes with the well established methodology of (backward) simulation diagrams to prove the semantic preservation sought.

In contrast, Xia et al. [3] recently introduced an alternative toolbox to define formal semantics of languages in the Coq proof assistant: the so-called interaction trees (itrees). Drawing on algebraic effects, itrees are used by parameterizing the semantics of the language by a signature of events, such as read and writes to a memory for instance. The semantics is then represented as a potentially infinite tree of interactions, through this signature of events, with the environment. The *effects* corresponding to the answers provided to these events by the environment is then implemented in a second phase, into an appropriate monad.

In contrast to the traditional small step approach to formal semantics, itrees provide tools to help defining semantics that satisfy three crucial properties: executability (the formal semantics can be run), compositionality (the semantics is a denotation, it is defined by recursion on the syntax) and modularity (each effect of the language is given a meaning in isolation). The approach scales to realist languages, as exampled in the formal semantics for LLVM IR developed by Zakowski et al. [5].

When it comes to proving the correctness of transformations of programs, interaction trees are equipped with a notion of weak bisimulation of programs [6]: the structure of the trees have to be the same in that events (e.g. reads and writes to memory) are matched one-to-one, but internal, non-observable computations, are ignored. The library of itrees comes with a rich structural equational theory to establish such simulations.

However, it provides no direct support to reason semantically about these events: we intuitively know that two consecutive writes to the same memory location should be equivalent to only performing the latter, but the bisimulation is too precise, it requires events to match one-to-one. In the current approach, we recover this kind of semantic equations by performing the reasoning after interpretation (i.e. implementation) of the event. This leads to proofs that are more specialized than need be, rendering the maintaining and evolution of large verified compilers harder.

This situation echoes the reminder administrated by Andrej Bauer [1]: "*What is algebraic about algebraic effects and handlers?*". The itrees are parameterized by the signature of events, but not by their theory! Through this internship, we propose to explore the resolution of this unsatisfactory situation.

LIP – UMR CNRS / ENS Lyon / UCB Lyon 1 / Inria – 69007 Lyon
E-mail : yannick.zakowski@ens-lyon.fr and ludovic.henrio@ens-lyon.fr

Location and supervision

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

Yannick Zakowski will act as main supervisor and is one of the two lead developers of the itree library, and the lead developer of its use in the Vellvm project. They therefore have experience both on the theoretical background and concrete Coq implementation side of the material. Ludovic Henrio has extensive expertise in the design of bisimulation for concurrent calculi.

Internship content

This internship relies on a fair amount of non-standard technical material. It is therefore expected for the intern to dedicate a consequent first part of the internship to familiarize themselves with this material. A previous familiarity with Coq or similar language is not mandatory, but would lighten the overhead for the intern.

More specifically, the internship would follow the following steps:

- Familiarization with Coq, the interaction trees and their implementation. The advisor and its collaborators have notably written a tutorial to this end.
- Familiarization with the relation of bisimulation at hand, and development of a precise understanding of the problem at hand by contrasting it to Andrej Bauer's (properly) algebraic formulation of algebraic effects.
- Design of an adequate data structure suitable to parameterize itrees by a theory, and definition of the new resulting bisimulation.
- Proof of the theory associated with the current bisimulation, over the new parameterized version.
- Design of toy examples demonstrating the benefit of this work: for instance, over a minimal language manipulating a memory, proof of a toy optimization independently from the implementation of the memory.

References

[1] Andrej Bauer. What is algebraic about algebraic effects and handlers? Technical report, 07 2018.

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

[3] 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.

[4] 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.

[5] 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.

[6] Yannick Zakowski, Paul He, Chung-Kil Hur, and Steve Zdancewic. An equational theory for weak bisimulation via generalized parameterized coinduction. In *Proceedings of the 9th Annual ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP '20, New York, NY, USA, 2020. ACM.