Formal verification of a static analyzer: abstract interpretation in type theory

Xavier Leroy

Inria Paris-Rocquencourt
xavier.leroy@inria.fr

(Joint work with David Pichardie, Sandrine Blazy, Jacques-Henri Jourdan, and Vincent Laporte.)

Abstract

Static analysis is the automatic inference and checking of simple properties of all executions of a program. Initially developed in the context of compilers to support code optimization, static analysis is very successful today for the formal verification of safety properties of critical software, owing to its good scalability. As is the case for all tools involved in the production and verification of critical software (compilers, code generators, program provers, model checkers), confidence in the results of a static analysis tool requires evidence that the tool is sound and correctly over-approximates all possible executions of the program. Such evidence can take the form of a soundness proof mechanized using a proof assistant [5, 4].

Abstract interpretation [2] is an elegant, powerful mathematical framework to define and reason about static analyses. In particular, it is not limited to so-called "non-relational" analyses (inferring properties of a single value or variable) and works naturally for "relational" analyses (inferring relations between several variables, such as linear inequalities). The classic presentation of abstract interpretation involves Galois connections. It has the advantage that, once the meaning of abstract data is chosen via a Galois connection, the abstract operators used by the static analyzer can, in principle, be derived systematically from the concrete semantics, in a way that is not only sound by construction, but also relatively optimal.

However, the theory of Galois connections is resolutely set-theoretical, involving non-computable functions and equational reasoning over set comprehensions, making it very hard to express in type theory and to use in a proof assistant such as Coq. To overcome this difficulty, Pichardie *et al* [6, 1] developed and mechanized an alternative presentation of abstract interpretation, using only the "γ" (concretization) part of Galois connections, viewed as relations "*concrete-datum* $\in$ *abstract-datum*". The calculational style is lost, and relative optimality is no longer guaranteed, but soundness proofs are easily conducted with a proof assistant.

In the context of the Verasco project, we are currently trying to scale Pichardie's approach all the way to the development and Coq verification of a realistic static analyzer based on abstract interpretation for the CompCert subset of the C language. Proper modular decomposition is crucial to build the appropriate abstractions as a hierarchy combining numerical and memory abstract domains. While the general interface of a non-relational domain is well known, giving such an interface for relational domains is more challenging, and so is formulating generic composition operators (such as reduced products) between such domains. Another enabling technique is the opportunistic use of validation a posteriori to obviate the need to prove complicated algorithms such as fixpoint iteration with widening and narrowing, or operations over polyhedra for relational domains of linear inequalities [3].

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

References

[1] David Cachera, Thomas Jensen, David Pichardie, and Vlad Rusu. Extracting a data flow analyser in constructive logic. *Theoretical Computer Science*, 342(1):56–78, 2005.

[2] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Fourth ACM Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM, 1977.

[3] Alexis Fouilhé, David Monniaux, and Michaël Périn. Efficient generation of correctness certificates for the abstract domain of polyhedra. In *Static Analysis - 20th International Symposium (SAS 2013)*, volume 7935 of *Lecture Notes in Computer Science*, pages 345–365. Springer, 2013.

[4] Paolo Herms, Claude Marché, and Benjamin Monate. A certified multi-prover verification condition generator. In *Verified Software: Theories, Tools, Experiments (VSTTE 2012)*, volume 7152 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2012.

[5] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[6] David Pichardie. *Interprétation abstraite en logique intuitionniste: extraction d'analyseurs Java certifiés*. PhD thesis, Université Rennes 1, 2005.