Comparing Techniques for Certified Static Analysis

David Cachera, David Pichardie

Comparing Techniques for Certified Static Analysis*

David Cachera†
ENS Cachan, IRISA,
Rennes, France

David Pichardie
INRIA, Centre Rennes - Bretagne Atlantique,
Rennes, France

Abstract

A certified static analysis is an analysis whose semantic validity has been formally proved correct with a proof assistant. The recent increasing interest in using proof assistants for mechanizing programming language metatheory has given rise to several approaches for certification of static analysis. We propose a panorama of these techniques and compare their respective strengths and weaknesses.

1 Introduction

Nowadays safety critical systems are validated through long and costly test campaigns. Static analysis is a promising complementary technique that allows to automatically prove the absence of restricted classes of bugs. A significant example is the state-of-the-art ASTRÉE static analyzer for C [12] which has proven some critical safety properties for the primary flight control software of the Airbus A340 fly-by-wire system. Taking note of such a success, the next question is: should we completely remove the test campaign dedicated to the same class of bugs? If we trust the result of the analyzer, of course the answer is yes, but should we trust it? The analyzer itself can be certified by testing, but exhaustivity cannot be achieved. In this paper, we show how mechanized proofs can be used to certify static analyzers or their results.

Abstract interpretation [11] is a general theory that aims at designing provably correct static analyzers, but pencil-and-paper proofs hardly scale to real-size analyzers for real-size programming languages. Proof assistants [9, 18, 19] allow to mechanically specify, program and prove correct static analyzers with respect to a formal model of the programming language semantics. If the feasibility of such a technique has been demonstrated for various kinds of analyses and programming languages [14, 2, 7, 21, 10, 5, 23], many approaches coexist and some of them differ in the kind of guarantee they give on the targeted static analysis. In this work, we make a comparison between different techniques, taking into account the proof effort, the obtained guarantee and maintenance problems. The paper is organized as follows: we first show how an analysis can be specified depending on the expected guarantees. We then address the question of computing a certified solution of the analysis. Finally we investigate the use of deductive verification to validate the invariant generated by an analysis.

For presentation purposes, all the analyzes we describe here will share a common semantical basis, which we give below, together with a summary of abstract interpretation principles. A program P is a graph (N,E) where $N \geq 1$ is the number of vertices (control points), and E a set of edges $(n,m) \in \{1..N\} \times \{1..N\}$. Each edge is labeled by an instruction $i_{n,m}$ from a set $\mathbb{I}$. Among nodes, we distinguish an entry point n_e such that there is no incoming edge in n_e. A *state* of a program P is composed of a control point n and an environment ρ: $\text{State} = \{1..N\} \times \text{Env}$. The *concrete semantics* of an instruction $i \in \mathbb{I}$ is given by a binary relation $\rightarrow_i$ over Env. The transfer function $F_i : \mathscr{P}(\text{Env}) \rightarrow \mathscr{P}(\text{Env})$ associated to instruction i is then defined by $F_i(S) = \{\rho' \mid \exists \rho \in S : \rho \rightarrow_i \rho'\}$. Given S_0 an initial set of environments, the collecting semantics $[\![P]\!]^c$ of P is the least solution $X \in \mathscr{P}(\text{Env})^N$ of $S_{n_e} = S_0$ and $\forall m \in \{1..N\}$, $S_m = \bigcup_{(n,m)\in E} F_{i_{n,m}}(S_n)$.

The abstract interpretation formalism gives us a way to over-approximate the solution to these equations. An abstract semantics is expressed w.r.t. an abstract domain[1] $\text{Env}^\sharp$ (usually a complete lattice

*Work partially supported by EU project MOBIUS, ANR-U3CAT grant and FNRAE ASCERT grant.

†Currently delegated as a full time researcher at INRIA, Centre Rennes - Bretagne Atlantique.

[1]Here we focus on the specific case where the abstract states are mappings from $\{1..N\}$ to $\text{Env}^\sharp$.

$(\text{Env}^\sharp, \sqsubseteq_\sharp)$), and the relation between concrete and abstract semantics is given by a Galois connection $(\alpha, \mathscr{P}(\text{Env}), \text{Env}^\sharp, \gamma)$, *i.e.* a pair of monotone mappings such that $\forall S \subseteq \text{Env} : S \subseteq \gamma(\alpha(S))$ and $\forall d \in \text{Env}^\sharp : \alpha(\gamma(d)) \sqsubseteq_\sharp d$. The *abstraction function* α maps a set S of environments to an abstract environment, which can be seen as the least property satisfied by all elements of S. The *concretization function* γ maps an abstract environment to all the concrete environments satisfying the corresponding property. We are interested in computing an abstract semantics $[\![P]\!]^\sharp : \{1..N\} \to \text{Env}^\sharp$ which is a certified *correct (over-)approximation* of the concrete collecting semantics, *i.e.* $\forall i \in \{1..N\} : [\![P]\!]^c(i) \subseteq \gamma([\![P]\!]^\sharp(i))$.

In addition to the systematic treatment of this safety issue, the theory provides an optimal specification of the abstract semantics: given a mapping $F : \mathscr{P}(\text{Env}) \to \mathscr{P}(\text{Env})$, a correct abstraction of F is a mapping $F^\sharp$ verifying $F \circ \gamma \subseteq \gamma \circ F^\sharp$, or equivalently $\alpha \circ F \circ \gamma \sqsubseteq_\sharp F^\sharp$. The case where $F^\sharp = \alpha \circ F \circ \gamma$ thus provides the best correct abstraction of F. The abstract semantics computed by the analyzer will then mimic the collecting semantics, in the sense that it also operates through abstract transfer functions $F_i^\sharp$. The result $[\![P]\!]^\sharp$ of the analysis is thus the least solution of

$$\alpha(S_0) \sqsubseteq_\sharp S^\sharp_{n_e} \quad \text{and} \quad \forall (n,m) \in E, F^\sharp_{i_{n,m}}(S^\sharp_n) \sqsubseteq_\sharp S^\sharp_m \tag{1}$$

and each $F_i^\sharp$ is proved an *optimal* correct abstraction of F_i. In order to save space, all properties like $\alpha(S_0) \sqsubseteq_\sharp S^\sharp_{n_e}$ dealing with the treatment of initial states will be discarded from the rest of this paper.

2 Analyzer Specification

Analysis soundness must be established with respect to the concrete semantics using a correctness relation that relates the concrete and the abstract domains. In this section we consider two formal frameworks that both enforce the following essential soundness property: any solution $S^\sharp$ of (1) is a correct approximation of $[\![P]\!]^c$. Various ways can be taken between these two opposite approaches, some of them have been investigated in [14, 7, 21, 10, 5].

2.1 Deep Analyzer Specification

This first approach (so far only experimented in [16]) exactly follows the Galois connection formalism by providing mechanized proofs of all the classical properties. We briefly recall the components of a static analyzer based on this formalism and make more precise the proof requirements.

component	**mathematical structure**	**properties to prove**
abstract domain	complete lattice $(\{1..N\} \to \text{Env}^\sharp, \dot{\sqsubseteq}^\sharp, \dot{\sqcup}^\sharp, \dot{\sqcap}^\sharp)$	existence of a lub
correctness relation	Galois connection $(\alpha, \mathscr{P}(\text{Env}), \text{Env}^\sharp, \gamma)$	Galois connection definition
abstract semantics	abstract transfer functions	$F_i^\sharp = \alpha \circ F_i \circ \gamma$

Defining the abstract domain as a complete lattice constrains us to provide a proof of existence of a least upper bound for any subset of abstract elements (or a least fixpoint for any monotone function). Moreover, from a proof assistant point of view, the $\sqcup^\sharp$ operator is not constructive, which hampers its implementation. Taking a Galois connection as a correctness criterion also increases the number of proofs to be done, since this also provides an optimality property.

Let us take an example to illustrate this point. We consider a numerical abstraction based on the domain of intervals. The concrete domain is $(\mathscr{P}(\mathbb{Z}), \subseteq, \cup, \cap)$, and the abstract domain is the lattice of intervals $\{[a,b] \mid a \in \mathbb{Z} \cup \{-\infty\}, b \in \mathbb{Z} \cup \{+\infty\}, a \leq b\} \cup \bot_{\text{Int}}$. The corresponding abstraction and concretization functions are defined by the following equations

$$\begin{array}{rcl} \gamma(\bot_{\text{Int}}) & = & \emptyset \\ \gamma([a,b]) & = & \{x \in \mathbb{Z} \mid a \leq x \leq b\} \end{array} \qquad \alpha(P) = \begin{cases} \bot_{\text{Int}} \text{ if } P = \emptyset \\ [\inf(P), \sup(P)] \text{ otherwise} \end{cases}$$

If we want to compute $\alpha \circ F \circ \gamma$ for a given computable function F, the use of α might force us to provide a proof that a given subset of $\mathbb{Z}$ is not bounded to ensure that $\top = [-\infty, +\infty]$ is a correct result of the analysis. If no optimality property were involved, $\top$ could be taken as a correct result in any case. As an example of abstract transfer function, let us now consider an operator $\times^\sharp$ intended to abstract the integer multiplication. Given intervals $[a,a']$ and $[b,b']$, let $[c,d] = [a,b] \times^\sharp [a',b']$. Soundness of the abstract operator means that $a \leq x \leq b$ and $a' \leq x' \leq b'$ imply $c \leq x \times x' \leq d$, while its optimality additionally ensures that $c = \inf\{x \times x' \mid a \leq x \leq b \text{ and } a' \leq x' \leq b'\}$ and $d = \sup\{x \times x' \mid a \leq x \leq b \text{ and } a' \leq x' \leq b'\}$.

2.2 Shallow Analyzer Specification

A second approach consists in focusing only on the soundness of the analysis, without considering the optimality issue. The choices made here are directly inspired from [13], where the authors describe the design of the ASTRÉE static analyzer.

component	**mathematical structure**	**properties to prove**
abstract domain	$(\text{Env}^\sharp, \sqsubseteq_\sharp, \sqcup^\sharp, \sqcap^\sharp)$	no requirements on $\sqsubseteq_\sharp, \sqcup^\sharp$ or $\sqcap^\sharp$
correctness relation	$\gamma : \text{Env}^\sharp \to \mathscr{P}(\text{Env})$	$\forall c,d \in \text{Env}^\sharp, c \sqsubseteq_\sharp d \Rightarrow \gamma(c) \subseteq \gamma(d)$
abstract semantics	abstract transfer functions	soundness: $F_i \circ \gamma \subseteq \gamma \circ F_i^\sharp$

Let us consider again the $\times^\sharp$ operator. Its definition could include the following statement: $[a,b] \times^\sharp \top = \top$, which is a correct approximation. A more precise one would be $[a,b] \times^\sharp \top = 0$ if $a = b = 0$, $\top$ otherwise, but could be missed by our analysis, since optimality is not required any more. Hence, if the shallow framework requires far less machine proofs than the deep framework, it ensures only a minimal amount of properties on the analysis which is doubtless sound but may still contain several precision bugs that are notoriously hard to debug.

3 Result Computation

The requirements made during the specification phase ensure that any solution of (1) is a correct approximation of $[\![P]\!]^c$, but do not specify *how* a solution is computed. One has to choose between various certification levels: from a complete proof of the whole analysis computation to a result-only certification.

3.1 Termination

If we aim at certifying the whole analyzer, we have to prove the termination of an algorithm computing a solution of (1). Termination proofs are known to be difficult and are seldom mechanized, or even precisely formalized. Even if we consider a complete lattice, the existence of a least fixpoint for monotone functions does not ensure the convergence of the computation in finite time. Except for the trivial case of finite height lattices, we have to exhibit specific properties of the domain, or to design operators that will ensure convergence.

A first approach consists in certifying that the lattice respects the ascending chain condition, *i.e.* that any increasing chain stabilizes in finite time. The main drawback of this approach is that it does not apply to popular abstract domains like intervals or polyhedra. A more common approach consists in designing widening and narrowing operators in order to accelerate the convergence [11]. The widening operator provides a guarantee to reach a post-fixpoint in finite time, while the narrowing operator is used to improve the precision of this post-fixpoint by a descending iteration. In both approaches, a mechanized proof of termination has to cope with constructivity issues, and the criteria of ascending

chain or termination of widening-based iteration have to be modified in consequence. In any case, a key issue for termination proofs is to provide modular lattice constructions, thus allowing for building a global proof out of basic blocks (usual numerical abstract domains for instance) [14, 20].

If one wants to avoid to perform tedious termination proofs, it is also possible to artificially bound the number of iterations in the abstract semantics computation [15]. In that case, one has to certify that any iteration yields a correct result, even if not the best one, or to check that the final result is indeed a correct approximation of the concrete semantics. This technique leads us to result certification.

3.2 Result-only Certification

In a safety-critical context, it is likely that the high confidence we are looking for is not for all results of the analyzer but for a few specific ones like those obtained for the next version of the flight-command program. Instead of globally certifying the analyzer itself, it might be interesting to provide a tool that checks the correctness of its result. This approach is directly related to the Proof Carrying Code technique [17, 1], where the code producer provides a formal proof that the code respects some safety requirements defined by the end-user. The user then verifies the proof with an automated and trusted checker. This use of a PCC technique for analysis certification has been proposed in [6]. The main requirement is the same as in the previous approaches: we still have to give a proof that $F_i \circ \gamma \subseteq \gamma \circ F_i^\sharp$. But now, instead of computing a solution of (1), we just have to check that a given certificate $S^\sharp$ is indeed a solution. Note that this proof can be automatically discharged by a computation.

4 Deductive Verification of Analysis Results

Since the main goal of static analysis is to generate invariants over program executions, it is natural to try to validate these invariants with deductive verification techniques that are traditionally applied for handwritten program invariants. In this section, we assume an axiomatic semantics given by a deductive judgment $\vdash \{\phi\}\ i\ \{\psi\}$ for each instruction $i \in \mathbb{I}$ such that, when property ϕ holds before executing i, ψ must hold after. Here, ϕ and ψ are formulas in a given logic language $\mathscr{L}$. We denote by $\pi : \{\phi\}i\{\psi\}$ a Hoare-proof derivation π that is a valid proof of $\vdash \{\phi\}\ i\ \{\psi\}$. We note $\rho \models \phi$ when an environment ρ satisfies property ϕ. To validate a set of invariants $\phi_1, \ldots, \phi_N$ attached to each control point, it is sufficient to provide a set of Hoare proofs $\pi_{n,m} : \{\phi_n\}\ i_{n,m}\ \{\phi_m\}$ for all (n,m) in E.

An analysis result has to be first transformed into a set of assertions in the language $\mathscr{L}$. We thus assume that each abstract element $a^\sharp \in \mathrm{Env}^\sharp$ can be translated into a formula $\ulcorner a^\sharp \urcorner$ in $\mathscr{L}$. For an interval analysis where we attach an interval to each variable of a program (restricted here to x and y), and for a first order language with arithmetic, any abstract element $\{x : [a,b], y : [b,c]\}$ will be translated into a formula like $a \leq \mathtt{x} \leq b \wedge c \leq \mathtt{y} \leq d$.
In this setting, an analysis result $S^\sharp$ is certified once the following statements have been formally machine checked.

Hoare logic soundness	$\vdash \{\phi\}\ i\ \{\psi\}$ implies $(\forall \rho, \rho', \rho \rightarrow_i \rho'$ and $\rho \models A$ implies $\rho' \models B)$
Provability of assertions	$\forall (n,m) \in E,\ \{\ulcorner S_n^\sharp \urcorner\}\ i_{n,m}\ \{\ulcorner S_m^\sharp \urcorner\}$is provable

The advantage of the approach is that the soundness of Hoare logic can be proved once and for all, and used for several static analyses. If we assume that the Hoare logic is not only sound but also complete, that transformation $\ulcorner \cdot \urcorner$ preserves satisfiability (*i.e.* for all $\rho \in \mathrm{Env}$ and $a^\sharp \in \mathrm{Env}^\sharp$, $\rho \in \gamma(a^\sharp)$ iff $\rho \models \ulcorner a^\sharp \urcorner$), and that each transfer function $F_i^\sharp$ is sound w.r.t. F_i, then the corresponding set of Hoare triples is provable for any solution $S^\sharp$ of the analysis. However, a proof of these triplets still has to be constructed, without entering a painful manual process for each of them. A first approach, proposed by Seo *et. al.* [22],

instruments the analyzer to make it produce a proof derivation $\pi_{n,m}$ for all edges (n,m). This approach has also been followed by Beringer *et. al.* [3, 4] who translate type derivations into Hoare proofs. The approach proposed independently by Chaieb [8] relies on a weakest precondition computation. This time the analyzer is instrumented to produce proof terms for the verification conditions generated by a weakest precondition computation. If these approaches are elegant, they remain difficult to implement because generating proof terms requires more technical ability than transposing a pencil-and-paper proof into a proof assistant.

Ideally, the proof obligations (obtained for example with the weakest precondition calculus) should be automatically discharged by a trustworthy theorem prover, hence the uselessness of generating proofs. However, each analysis may require specific decision procedures. For example, in the interval analysis the verification condition for the Hoare triplet $\{\ulcorner\{x : [1,2]\ y : [3,4]\}\urcorner\}\ x := x * y\ \{\ulcorner\{x : [3,8]\ y : \top\}\urcorner\}$ will be $(1 \leq \mathtt{x} \leq 2 \wedge 3 \leq \mathtt{y} \leq 4) \Rightarrow (3 \leq \mathtt{x} \times \mathtt{y} \leq 8)$ which does not fall in the Presburger arithmetic fragment.nstead of producing a specific proof for each verification condition, we believe it may be a better idea to strengthen the theorem prover with a decision procedure able to discharge exactly this kind of formula. In addition to validating the analysis result, it is likely to improve the prover itself. Since each analyzer addresses dedicated decision procedures in its transfer functions and partial order tests, it would be useful to share this capability with an automatic prover. The research line we advocate here is then to design abstract domains and decision procedures in parallel. To ensure the validity of the approach, the decision procedures must themselves be certified, *i.e.* must generate proof terms of validity.

5 Conclusion

We have considered several techniques for certifying the soundness of a static analyzer or of its result. Of course, there is no *silver-bullet* technique: if the deep approach is the most greedy in terms of proof effort, it is the only one that detects precision bugs. Revealing such bugs too late during a validation campaign may compromise the availability of the safety critical system that has to be validated in due time. Deductive verification appears as a promising technique but its apparent generality must be tempered: the underlying logic is not always expressive enough to translate the result of the analysis and automatically discharging verification conditions requires technical instrumentation of the analyzer. Building abstract domains and decision procedures in parallel seems an interesting research line to push further.

References

[1] E. Albert, G. Puebla, and M. Hermenegildo. Abstraction-carrying code. In *In Proc. of LPAR'04*, pages 380–397. Springer LNAI vol 3452, 2005.

[2] G. Barthe, G. Dufay, L. Jakubiec, and S. Melo de Sousa. A formal correspondence between offensive and defensive javacard virtual machines. In *VMCAI*, pages 32–45. Springer-Verlag LNCS vol. 2294, 2002.

[3] L. Beringer, M. Hofmann, A. Momigliano, and O. Shkaravska. Automatic certification of heap consumption. In *Proc. of LPAR 2004*, pages 347–362. Springer LNCS vol. 3452, 2004.

[4] L. Beringer, M. Hofmann, and M. Pavlova. Certification using the Mobius base logic. In *Proc. of FMCO'07*, pages 25–51. Springer LNCS vol. 5382, 2007.

[5] Y. Bertot, B. Grégoire, and X. Leroy. A structured approach to proving compiler optimizations based on dataflow analysis. In *Proc. of TYPES'04*, pages 66–81. Springer LNCS vol. 3839, 2006.

[6] F. Besson, T. Jensen, and D. Pichardie. Proof-Carrying Code from certified abstract interpretation to fixpoint compression. *Theoretical Computer Science*, 364(3):273–291, 2006.

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

[8] A. Chaieb. Proof-producing program analysis. In *Proc. of ICTAC 2006*. Springer LNCS vol. 4281, 2006.

[9] The Coq proof assistant. `http://coq.inria.fr/`.

[10] S. Coupet-Grimal and W. Delobel. A uniform and certified approach for two static analyses. In *Proc. of TYPES'04*, pages 115–137. Springer LNCS vol. 3839, 2006.

[11] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL'77*, pages 238–252. ACM Press, 1977.

[12] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE analyser. In *Proc. of ESOP'05*, pages 21–30. Springer LNCS vol. 3444, 2005.

[13] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Combination of abstractions in the ASTRÉE static analyzer. In *Proc. of ASIAN'06*. Springer LNCS vol. 4435, 2007.

[14] G. Klein and T. Nipkow. A machine-checked model for a Java-like language, virtual machine and compiler. *ACM Transactions on Programming Languages and Systems*, 28(4):619–695, 2006.

[15] X. Leroy. Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. In *Proc. of POPL'06*, pages 42–54. ACM Press, 2006.

[16] D. Monniaux. Réalisation mécanisée d'interpréteurs abstraits. Rapport de DEA, Université Paris VII, 1998.

[17] G. C. Necula. Proof-carrying code. In *Proc. of POPL'97*, pages 106–119. ACM Press, 1997.

[18] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer-Verlag, 2002.

[19] S. Owre, J. M. Rushby, and N. Shankar. PVS: a prototype verification system. In *Proc. of CADE'92*, pages 748–752. Springer LNCS vol. 607, 1992.

[20] D. Pichardie. Building certified static analysers by modular construction of well-founded lattices. In *Proc. of FICS'08*, volume 212 of *Electronic Notes in Theoretical Computer Science*, pages 225–239, 2008.

[21] A. Salcianu and K. Arkoudas. Machine-Checkable Correctness Proofs for Intra-procedural Dataflow Analyses. In *Proc. of COCV'05*, pages 53–68. Elsevier ENTCS vol. 141:2, 2005.

[22] S. Seo, H. Yang, and K. Yi. Automatic construction of Hoare proofs from abstract interpretation results. In *Proc. of APLAS 2003*, pages 230–245. Springer, 2003.

[23] M. Wildmoser, A. Chaieb, and T. Nipkow. Bytecode analysis for proof carrying code. In *Proc. of Bytecode'05*, pages 19–34. Elsevier ENTCS vol. 141:1, 2005.