Formal verification by abstract interpretation

CONFERENCE PAPER · APRIL 2012

DOI: 10.1007/978-3-642-28891-3_3

Patrick Cousot
New York University

Formal Verification by Abstract Interpretation

Patrick Cousot

CIMS NYU, New York, USA & CNRS–ENS–INRIA, Paris, France

Abstract. We provide a rapid overview of the theoretical foundations and main applications of abstract interpretation and show that it currently provides scaling solutions to achieving assurance in mission- and safety-critical systems through verification by fully automatic, semantically sound and precise static program analysis.

Keywords: Abstract interpretation, Abstraction, Aerospace, Certification, Cyber-physical system, Formal Method, Mission-critical system, Runtime error, Safety-critical system, Scalability, Soundness, Static Analysis, Validation, Verification.

1 Abstract interpretation

Abstract interpretation [9–13] is a theory of abstraction and constructive approximation of the mathematical structures used in the formal description of programming languages and the inference or verification of undecidable program properties.

The design of an inference or verification method by abstract interpretation starts with the formal definition of the semantics of a programming language (formally describing all possible program behaviors in all possible execution environments), continues with the formalization of program properties, and the expression of the strongest program property of interest in fixed point form.

The theory provides property and fixed point abstraction methods than can be constructively applied to obtain formally verified abstract semantics of the programming languages where, ideally, only properties relevant to the considered inference or verification problem are preserved while all others are abstracted away.

Formal proof methods for verification are derived by checking fixed point by induction. For property inference in static analyzers, iterative fixed point approximation methods with convergence acceleration using widening/narrowing provide effective algorithms to automatically infer abstract program properties (such as invariance or definite termination) which can then be used for program verification by fixed point checking.

Because program verification problems are undecidable for infinite systems, any fully automatic formal method will fail on infinitely many programs and, fortunately, also succeed on infinitely many programs. An abstraction over-approximates the set of possible concrete executions and so may include executions not existing in the concrete. This is not a problem when such fake executions do not affect the property to be verified (e.g. for invariance the execution time is irrelevant). Otherwise this may cause a false alarm in that the property is violated by an inexistent execution. In this case, the abstraction must be refined to better distinguish between actual and fake program executions.

To maximize success for specific applications of the theory, it is necessary to adapt the abstractions/approximations so as to eliminate false alarms (when the analysis is too imprecise to provide a definite answer to the verification problem) at a reasonable cost. The choice of an abstraction which is precise enough to check for specified properties and imprecise enough to be scalable to very large programs is difficult. This can be done by refining or coarsening general-purpose abstractions.

A convenient way to adjust the precision/cost ratio of a static analyser consists in organizing the effective abstract fixed point computation in an abstract interpreter (mainly dealing with control) parameterized by abstract domains (mainly dealing with data). These abstract domains algebraically describe classes of properties and the associated logical operations, extrapolation operators (widening and narrowing needed to over-approximate fixed points) and primitive transformers corresponding to basic operations of the programming language (such as assignment, test, call, etc).

To achieve the desired precision, the various abstract domains can combined by the abstract interpreter, e.g. with a reduced product [28], so as to eliminate false alarms at a reasonable cost.

Several surveys of abstract interpretation [1, 7, 19, 21] describe this general methodology in more details.

2 A few applications of abstract interpretation

Abstract interpretation has applications in the syntax [22], semantics [14], and proof [20] of programming languages where abstractions are sound (no possible case is ever omitted in the abstraction) and complete (the abstraction is precise enough to express/verify concrete program properties in the abstract without any false alarm) but in general incomputable (but with severe additional hypotheses such as finiteness). Full automation of the verification task requires further sound but incomplete abstractions as applied to static analysis [9, 30], contract inference [27], type inference [6], termination inference [23] model-checking [8, 15, 16], abstraction refinement [29], program transformation [17] (including watermarking [18]), combination of decision procedures [28], etc.

3 Applications to assurance in mission- and safety-critical systems

Abstract interpretation has been successful this last decade in program verification for mission- and safety-critical systems. Significant applications of abstract interpretation to aerospace systems include e.g. airplane control-command [31, 34, 35] and autonomous rendezvous and docking for spacecraft [5].

An example is Astrée [1–4, 24–26] (`www.astree.ens.fr`) which is a static analyzer to verify the absence of runtime errors in structured, very large C programs with complex memory usages, and involving complex boolean as well as floating-point computations (which are handled precisely and safely by taking all possible rounding errors into account), but without recursion or dynamic memory allocation. Astrée targets embedded applications as found in earth transportation, nuclear energy, medical instrumentation, aeronautics and space flight, in particular synchronous control/command such as electric flight control.

Astrée reports any division by zero, out-of-bounds array indexing, erroneous pointer manipulation and dereferencing (null, uninitialized and dangling pointers), integer and floating-point arithmetic overflow, violation of optional user-defined assertions to prove additional run-time properties (similar to assert diagnostics), code it can prove to be unreachable under any circumstances (note that this is not necessarily all unreachable code due to over-approximations), read access to uninitialized variables. Astrée offers powerful annotation mechanisms, which enable the user to make external knowledge available to Astrée, or to selectively influence the analysis precision for individual loops or data structures. Detailed messages and an intuitive GUI help the user understand alarms about potential errors. Then, true runtime errors can be fixed, or, in case of a false alarm, the analyzer can be tuned to avoid them. These mechanisms allow to perform analyses with very few or even zero false alarms. Astrée is industrialised by AbsInt (`www.absint.com/astree`).

AstréeA [32, 33] is built upon Astrée to prove the absence of runtime errors and data races in parallel programs. Asynchrony introduces additional difficulties due to the semantics of parallelism (such as the abstraction of process interleaving, explicit process scheduling, shared memory model, etc).

4 Conclusion

Abstract interpretation has a broad spectrum of applications from theory to practice. Abstract interpretation-based static analysis is automatic, sound, scalable to industrial size software, precise, and commercially supported for proving the absence of runtime errors. It is a premium formal method to complement dynamic testing as recommended by DO-178C/ED-12C (`http://www.rtca.org/doclist.asp`).

Work supported in part by the CMACS NSF Expeditions in Computing award 0926166 and the European ARTEMIS project MBAT.

References

1. J. Bertrane, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Static analysis and verification of aerospace software by abstract interpretation. In *AIAA Infotech@Aerospace 2010*, Atlanta, Georgia, 20–22 April 2010. American Institute of Aeronautics and Astronautics.
2. J. Bertrane, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Static analysis by abstract interpretation of embedded critical software. *ACM SIGSOFT Software Engineering Notes*, 36(1):1–8, 2011.
3. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. In T. Æ. Mogensen, D. A. Schmidt, and I. H. Sudborough, editors, *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones [on occasion of his 60th birthday]*, volume 2566 of *Lecture Notes in Computer Science*, pages 85–108. Springer, 2002.

A. Goodloe and S. Person (Eds.): NFM 2012, LNCS 7226, pp. 3–7, 2012

4. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation 2003, San Diego, California, USA, June 9-11, 2003*, pages 196–207. ACM, 2003.
5. O. Bouissou, E. Conquet, P. Cousot, R. Cousot, J. Feret, K. Ghorbal, ric Goubault, D. Lesens, L. Mauborgne, A. Miné, S. Putot, X. Rival, and M. Turin. Space software validation using abstract interpretation. In *Proc. of the Int. Space System Engineering Conf., Data Systems in Aerospace (DASIA 2009)*, volume SP-669, pages 1–7, Istambul, Turkey, May 2009. ESA.
6. P. Cousot. Types as abstract interpretations. In *POPL*, pages 316–331, 1997.
7. P. Cousot. The calculational design of a generic abstract interpreter. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*. NATO ASI Series F. IOS Press, Amsterdam, 1999.
8. P. Cousot. Partial completeness of abstract fixpoint checking. In B. Y. Choueiry and T. Walsh, editors, *Abstraction, Reformulation, and Approximation, 4th International Symposium, SARA 2000, Horseshoe Bay, Texas, USA, July 26-29, 2000, Proceedings*, volume 1864 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2000.
9. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1977, Los Angeles, California, USA, January 17-19, 1977*, pages 238–252, 1977.
10. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1979, San Antonio, Texas, USA, January 17-19, 1979*, pages 269–282, 1979.
11. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *J. Log. Program.*, 13(2&3):103–179, 1992.
12. P. Cousot and R. Cousot. Abstract interpretation frameworks. *J. Log. Comput.*, 2(4):511–547, 1992.
13. P. Cousot and R. Cousot. Comparing the galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Programming Language Implementation and Logic Programming, 4th International Symposium, PLILP'92, Leuven, Belgium, August 26-28, 1992, Proceedings*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295. Springer, 1992.
14. P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *POPL*, pages 83–94, 1992.
15. P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Autom. Softw. Eng.*, 6(1):69–95, 1999.
16. P. Cousot and R. Cousot. Temporal abstract interpretation. In *Proceedings of the 4th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2000, Boston, Massachusetts, USA, January 19-21, 2000*, pages 12–25, 2000.
17. P. Cousot and R. Cousot. Systematic design of program transformation frameworks by abstract interpretation. In *POPL*, pages 178–190, 2002.
18. P. Cousot and R. Cousot. An abstract interpretation-based framework for software watermarking. In N. D. Jones and X. Leroy, editors, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, pages 173–185. ACM, 2004.
19. P. Cousot and R. Cousot. Basic concepts of abstract interpretation. In R. Jacquard, editor, *Building the Information Society*, pages 359–366. Kluwer Academic Publishers, 2004.
20. P. Cousot and R. Cousot. Bi-inductive structural semantics. *Inf. Comput.*, 207(2):258–283, 2009.
21. P. Cousot and R. Cousot. A gentle introduction to formal verification of computer systems by abstract interpretation. In J. Esparza, O. Grumberg, and M. Broy, editors, *Logics and Languages for Reliability and Security*, NATO Science Series III: Computer and Systems Sciences, pages 1–29. IOS Press, 2010.
22. P. Cousot and R. Cousot. Grammar semantics, analysis and parsing by abstract interpretation. *Theor. Comput. Sci.*, 412(44):6135–6192, 2011.

23. P. Cousot and R. Cousot. An abstract interpretation framework for termination. In J. Field and M. Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 245–258. ACM, 2012.
24. P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The Astrée analyzer. In S. Sagiv, editor, *Programming Languages and Systems, 14th European Symposium on Programming,ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3444 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 2005.
25. P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Why does Astrée scale up? *Formal Methods in System Design*, 35(3):229–264, 2009.
26. P. Cousot, R. Cousot, J. Feret, A. Miné, L. Mauborgne, D. Monniaux, and X. Rival. Varieties of static analyzers: A comparison with Astrée. In *First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, TASE 2007, June 5-8, 2007, Shanghai, China*, pages 3–20. IEEE Computer Society, 2007.
27. P. Cousot, R. Cousot, and F. Logozzo. Precondition inference from intermittent assertions and application to contracts on collections. In R. Jhala and D. A. Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 150–168. Springer, 2011.
28. P. Cousot, R. Cousot, and L. Mauborgne. The reduced product of abstract domains and the combination of decision procedures. In M. Hofmann, editor, *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 456–472. Springer, 2011.
29. P. Cousot, P. Ganty, and J.-F. Raskin. Fixpoint-guided abstraction refinements. In H. R. Nielson and G. Filé, editors, *Static Analysis, 14th International Symposium, SAS 2007, Kongens Lyngby, Denmark, August 22-24, 2007, Proceedings*, volume 4634 of *Lecture Notes in Computer Science*, pages 333–348. Springer, 2007.
30. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1978, Tucson, Arizona, USA, January 23-25, 1978*, pages 84–96, 1978.
31. D. Delmas and J. Souyris. Astrée: From research to industry. In H. R. Nielson and G. Filé, editors, *Static Analysis, 14th International Symposium, SAS 2007, Kongens Lyngby, Denmark, August 22-24, 2007, Proceedings*, volume 4634 of *Lecture Notes in Computer Science*, pages 437–451. Springer, 2007.
32. A. Miné. Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics. In *ACM SIGPLAN/SIGBED Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES'06)*, pages 54–63. ACM Press, June 2006.
33. A. Miné. Static analysis of run-time errors in embedded critical parallel c programs. In G. Barthe, editor, *Programming Languages and Systems - 20th European Symposium on Programming, ESOP 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6602 of *Lecture Notes in Computer Science*, pages 398–418. Springer, 2011.
34. J. Souyris. Industrial experience of abstract interpretation-based static analyzers. In R. Jacquart, editor, *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22-27 August 2004, Toulouse, France*, pages 393–400. Kluwer, 2004.
35. J. Souyris and D. Delmas. Experimental assessment of Astrée on safety-critical avionics software. In F. Saglietti and N. Oster, editors, *Computer Safety, Reliability, and Security, 26th International Conference, SAFECOMP 2007, Nuremberg, Germany, September 18-21, 2007*, volume 4680 of *Lecture Notes in Computer Science*, pages 479–490. Springer, 2007.

A. Goodloe and S. Person (Eds.): NFM 2012, LNCS 7226, pp. 3–7, 2012