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 [9,10,11,12,13] 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 [1,2,3,4,24,25,26] (`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`).

References

1. Bertrane, J., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Rival, X.: Static analysis and verification of aerospace software by abstract interpretation. In: AIAA Infotech@Aerospace 2010, Atlanta, Georgia, April 20-22. American Institute of Aeronautics and Astronautics (2010)
2. Bertrane, J., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Rival, X.: Static analysis by abstract interpretation of embedded critical software. ACM SIGSOFT Software Engineering Notes 36(1), 1–8 (2011)

3. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software. In: Mogensen, T.Æ., Schmidt, D.A., Sudborough, I.H. (eds.) The Essence of Computation. LNCS, vol. 2566, pp. 85–108. Springer, Heidelberg (2002)
4. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: 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, pp. 196–207. ACM (2003)
5. Bouissou, O., Conquet, E., Cousot, P., Cousot, R., Feret, J., Ghorbal, K., Goubault, E., Lesens, D., Mauborgne, L., Miné, A., Putot, S., Rival, X., Turin, M.: Space software validation using abstract interpretation. In: Proc. of the Int. Space System Engineering Conf., Data Systems in Aerospace (DASIA 2009), Istambul, Turkey, vol. SP-669, pp. 1–7. ESA (May 2009)
6. Cousot, P.: Types as abstract interpretations. In: POPL, pp. 316–331 (1997)
7. Cousot, P.: The calculational design of a generic abstract interpreter. In: Broy, M., Steinbrüggen, R. (eds.) Calculational System Design. NATO ASI Series F. IOS Press, Amsterdam (1999)
8. Cousot, P.: Partial Completeness of Abstract Fixpoint Checking. In: Choueiry, B.Y., Walsh, T. (eds.) SARA 2000. LNCS (LNAI), vol. 1864, pp. 1–25. Springer, Heidelberg (2000)
9. Cousot, P., Cousot, R.: 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, pp. 238–252 (1977)
10. Cousot, P., Cousot, R.: 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, pp. 269–282 (1979)
11. Cousot, P., Cousot, R.: Abstract interpretation and application to logic programs. J. Log. Program. 13(2&3), 103–179 (1992)
12. Cousot, P., Cousot, R.: Abstract interpretation frameworks. J. Log. Comput. 2(4), 511–547 (1992)
13. Cousot, P., Cousot, R.: Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In: Bruynooghe, M., Wirsing, M. (eds.) PLILP 1992. LNCS, vol. 631, pp. 269–295. Springer, Heidelberg (1992)
14. Cousot, P., Cousot, R.: Inductive definitions, semantics and abstract interpretation. In: POPL, pp. 83–94 (1992)
15. Cousot, P., Cousot, R.: Refining model checking by abstract interpretation. Autom. Softw. Eng. 6(1), 69–95 (1999)
16. Cousot, P., Cousot, R.: 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, pp. 12–25 (2000)
17. Cousot, P., Cousot, R.: Systematic design of program transformation frameworks by abstract interpretation. In: POPL, pp. 178–190 (2002)
18. Cousot, P., Cousot, R.: An abstract interpretation-based framework for software watermarking. In: Jones, N.D., Leroy, X. (eds.) Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, pp. 173–185. ACM (2004)

19. Cousot, P., Cousot, R.: Basic concepts of abstract interpretation. In: Jacquard, R. (ed.) Building the Information Society, pp. 359–366. Kluwer Academic Publishers (2004)
20. Cousot, P., Cousot, R.: Bi-inductive structural semantics. Inf. Comput. 207(2), 258–283 (2009)
21. Cousot, P., Cousot, R.: A gentle introduction to formal verification of computer systems by abstract interpretation. In: Esparza, J., Grumberg, O., Broy, M. (eds.) Logics and Languages for Reliability and Security. NATO Science Series III: Computer and Systems Sciences, pp. 1–29. IOS Press (2010)
22. Cousot, P., Cousot, R.: Grammar semantics, analysis and parsing by abstract interpretation. Theor. Comput. Sci. 412(44), 6135–6192 (2011)
23. Cousot, P., Cousot, R.: An abstract interpretation framework for termination. In: Field, J., Hicks, M. (eds.) Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, pp. 245–258. ACM (2012)
24. Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: The ASTREÉ Analyzer. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 21–30. Springer, Heidelberg (2005)
25. Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Rival, X.: Why does Astrée scale up? Formal Methods in System Design 35(3), 229–264 (2009)
26. Cousot, P., Cousot, R., Feret, J., Miné, A., Mauborgne, L., Monniaux, D., Rival, X.: Varieties of static analyzers: A comparison with Astrée. In: First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, TASE 2007, Shanghai, China, June 5-8, pp. 3–20. IEEE Computer Society (2007)
27. Cousot, P., Cousot, R., Logozzo, F.: Precondition Inference from Intermittent Assertions and Application to Contracts on Collections. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 150–168. Springer, Heidelberg (2011)
28. Cousot, P., Cousot, R., Mauborgne, L.: The Reduced Product of Abstract Domains and the Combination of Decision Procedures. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 456–472. Springer, Heidelberg (2011)
29. Cousot, P., Ganty, P., Raskin, J.-F.: Fixpoint-Guided Abstraction Refinements. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 333–348. Springer, Heidelberg (2007)
30. Cousot, P., Halbwachs, N.: 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, pp. 84–96 (1978)
31. Delmas, D., Souyris, J.: Astrée: From Research to Industry. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 437–451. Springer, Heidelberg (2007)
32. Miné, A.: 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 2006), pp. 54–63. ACM Press (June 2006)
33. Miné, A.: Static Analysis of Run-Time Errors in Embedded Critical Parallel C Programs. In: Barthe, G. (ed.) ESOP 2011. LNCS, vol. 6602, pp. 398–418. Springer, Heidelberg (2011)
34. Souyris, J.: Industrial experience of abstract interpretation-based static analyzers. In: Jacquart, R. (ed.) IFIP 18th World Computer Congress, Topical Sessions, Building the Information Society, Toulouse, France, August 22-27, pp. 393–400. Kluwer (2004)
35. Souyris, J., Delmas, D.: Experimental Assessment of Astrée on Safety-Critical Avionics Software. In: Saglietti, F., Oster, N. (eds.) SAFECOMP 2007. LNCS, vol. 4680, pp. 479–490. Springer, Heidelberg (2007)