On Various Abstract Understandings of Abstract Interpretation

Patrick COUSOT
Courant Institute of Mathematical Sciences
New York University
pcousot@cims.nyu.edu, cims.nyu.edu/˜pcousot

Abstract—**We discuss several possible understandings and misunderstandings of Abstract Interpretation theory and practice at various levels of abstraction.**

Keywords-**Abstract Interpretation, Abstraction, Completeness, Formal methods, Semantics, Semantics, Soundness, Static analysis, Verification.**

ABSTRACT INTERPRETATION FOR STATIC ANALYSIS

Abstract Interpretation (see [10], [11] for gentle introductions) can be, and is often, understood in a very narrow sense: an algorithm for static analysis of sequential programs with widening and narrowing, even maybe restricted to interval analysis only.

This was indeed the origin of the concept [6] and the very first fully automatic infinitary static analysis, rapidly followed by more expressive and costly relational analyzes [15].

For programming languages, *i.e.* infinitely many programs, static analysis with infinitary abstractions and widening/narrowing is always terminating. It is also strictly more powerful than finite abstractions, including finite abstractions with refinements (which are often are allowed not to terminate) [9].

This specific static analysis algorithm is the very minimal view of Abstract Interpretation necessary to understand how production-quality static analyzers like ASTRÉE do operate [14] and why it scales up with high precision for domain-specific applications including for parallel programs [20] (but obviously not for all programs out of its application domain).

This algorithmic view is insufficient to understand why static analyzers produce credible information about program executions.

By credible, we understand either correct (which is unfortunately not the case of most static analyzers, which are often incorrect) or with a definite explanation of potential incorrectnesses [1]. For example ASTRÉE reports all potential buffer overruns. But the analysis covers only all prefix executions prior to the very first such buffer overrun, if any (because the program behavior is completely unpredictable after a buffer overrun, including because of the possible destruction of the executed code).

CONSTRUCTION OF ABSTRACTIONS BY ABSTRACT INTERPRETATION

A broader acceptation of Abstract Interpretation includes this soundness problematics. To ensure that the static analysis is correct/sound for all programs of a programming language, it is necessary to compare the results of the static analysis with a formal definition of the program semantics (sometimes called a model) [7].

Abstract Interpretation goes much further by showing how to formally construct the static analyzer from the definition of program properties as specified by the semantics [8].

These ideas lead to mechanically checked static analyzers [19], and hopefully in the future, to mechanically constructed static analyzers (as has been done by hand, *e.g.* in [3] for type systems).

HIERARCHIES OF ABSTRACTIONS

A more profound understanding of Abstract Interpretation leads to a much broader scope of application. To cope with complex problems, *e.g.* undecidable ones, it is necessary to abstract the structure of the concrete space on which this problem is defined into an abstract domain in which the problem is more tractable. This concrete structure is usually a concrete domain plus operations on that concrete domain such as transformers and fixpoint definitions.

Interestingly, the abstraction of the domain of properties of the concrete space induces the abstraction of properties of operations of the structure (but for extrapolation and interpolation operators such as widening and narrowing which are orthogonal approximation concepts for convergence acceleration of iterative computations *e.g.* of fixpoints).

Given a semantics, all possible abstractions form a hierarchy formalizing all possible ways of reasoning on programs in the abstract (called the lattice of abstract interpretations [8, Sect. 8]).

COMPLETENESS OF ABSTRACTIONS

The completeness question, is whether solving the problem in the abstract is always possible.

A common misunderstanding is to claim that Abstract Interpretation is incomplete by nature.

There are indeed many examples of complete abstractions from the FIRST algorithm abstracting the language defined by a context free program [12] to hierarchies of semantics (operational, denotational, axiomatic, *etc*) [4].

Any abstraction can indeed be always refined to a complete one [18] (so there is always a most abstract refinement of an abstraction to make a proof) and symmetrically any abstraction can always be simplified to achieve the same goals [17].

Incompleteness necessarily appears for undecidable problems for which all algorithms will ultimately fail on infinitely many counter-examples (including by not terminating).

Incomputability also appears for decidable problems with very high complexity leading to the combinatorial explosion of enumerative methods. The combinatorial explosion problem has never been solved [2], except by various forms of Abstract Interpretation. One can claim that the answer might be approximate but it is always correct and obtained in finite time which is better than a model-checker that run out of memory or a prover that times out with no clue at all on the problem to be solved.

Scope of Abstract Interpretation Theory

The usefulness of a theory lies in its capacity to explain a broad range of phenomena. In that respect, Abstract Interpretation can easily explain recent program verification methods, such as those based on Craig interpolation, that did not exist at the time the theory was elaborated. Abstract Interpretation even leads to useful generalizations. The key idea is the abstraction of mathematical induction [5].

At a time where verification techniques tend to be atomized into a multiplicity of ad-hoc methods, often applicable to a few well-chosen tiny programs, and requiring clues from the programmer that are tantamount to solving the problem by hand, Abstract Interpretation can play the rôle of a unifying theory.

A unifying theory of formal methods is necessary to get a global understanding and explanation of a vast and parcellelized research field.

In Abstract Interpretation, everything reduces to the understanding of a semantics, an abstraction into an abstract domain, and extrapolation/interpolation operators. This global understanding is much simpler than a wired algorithm, often introducing extra imprecisions.

Of course this does not diminish in any way the merits and originality of new ideas, that is of unexplored abstractions [13].

Acknowledgements: Work supported in part by NSF Grant CNS-1446511.

References

[1] Maria Christakis, Peter Müller, and Valentin Wüstholz. An experimental evaluation of deliberate unsoundness in a static program analyzer. In D'Souza et al. [16], pages 336–354.

[2] Edmund M. Clarke. My 27-year quest to overcome the state explosion problem. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, page 3. IEEE Computer Society, 2009.

[3] Patrick Cousot. Types as abstract interpretations. In Peter Lee, Fritz Henglein, and Neil D. Jones, editors, *Conference Record of POPL'97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, Paris, France, 15-17 January 1997*, pages 316–331. ACM Press, 1997.

[4] Patrick Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theor. Comput. Sci.*, 277(1-2):47–103, 2002.

[5] Patrick Cousot. Abstracting induction by extrapolation and interpolation. In D'Souza et al. [16], pages 19–42.

[6] Patrick Cousot and Radhia Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.

[7] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252. ACM, 1977.

[8] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In Alfred V. Aho, Stephen N. Zilles, and Barry K. Rosen, editors, *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979*, pages 269–282. ACM Press, 1979.

[9] Patrick Cousot and Radhia Cousot. Comparing the galois connection and widening/narrowing approaches to abstract interpretation. In Maurice Bruynooghe and Martin 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.

[10] Patrick Cousot and Radhia Cousot. Basic concepts of abstract interpretation. In René Jacquart, editor, *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22-27 August 2004, Toulouse, France*, volume 156 of *IFIP*, pages 359–366. Kluwer/Springer, 2004.

[11] Patrick Cousot and Radhia Cousot. A gentle introduction to formal verification of computer systems by abstract interpretation. In Javier Esparza, Bernd Spanfelner, and Orna Grumberg, editors, *Logics and Languages for Reliability and Security*, volume 25 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 1–29. IOS Press, 2010.

[12] Patrick Cousot and Radhia Cousot. Grammar semantics, analysis and parsing by abstract interpretation. *Theor. Comput. Sci.*, 412(44):6135–6192, 2011.

[13] Patrick Cousot and Radhia Cousot. Abstract interpretation: past, present and future. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 2:1–2:10. ACM, 2014.

[14] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival. Why does astrée scale up? *Formal Methods in System Design*, 35(3):229–264, 2009.

[15] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In Alfred V. Aho, Stephen N. Zilles, and Thomas G. Szymanski, editors, *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978*, pages 84–96. ACM Press, 1978.

[16] Deepak D'Souza, Akash Lal, and Kim Guldstrand Larsen, editors. *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, volume 8931 of *Lecture Notes in Computer Science*. Springer, 2014.

[17] Roberto Giacobazzi and Francesco Ranzato. Correctness kernels of abstract interpretations. *Inf. Comput.*, 237:187–203, 2014.

[18] Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.

[19] Jacques-Henri Jourdan, Vincent Laporte, Sandrine Blazy, Xavier Leroy, and David Pichardie. A formally-verified C static analyzer. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 247–259. ACM, 2015.

[20] Antoine Miné. Relational thread-modular static value analysis by abstract interpretation. In Kenneth L. McMillan and Xavier Rival, editors, *Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings*, volume 8318 of *Lecture Notes in Computer Science*, pages 39–58. Springer, 2014.