Abstract Interpretation: Theory and Practice

Patrick Cousot

École normale supérieure
Département d'informatique,
45 rue d'Ulm
75230 Paris cedex 05, France
Patrick.Cousot@ens.fr
http://www.di.ens.fr/~cousot/

Our objective in this talk is to give an intuitive account of abstract interpretation theory [1,2,3,4,5] and to present and discuss its main applications [6].

Abstract interpretation theory formalizes the conservative approximation of the semantics of hardware and software computer systems. The *semantics* provides a formal model describing all possible behaviors of a computer system in interaction with any possible environment. By *approximation* we mean the observation of the semantics at some level of abstraction, ignoring irrelevant details. *Conservative* means that the approximation can never lead to an erroneous conclusion.

Abstract interpretation theory provides *thinking tools* since the idea of abstraction by conservative approximation is central to reasoning (in particular on computer systems) and *mechanical tools* since the idea of an effective computable approximation leads to a systematic and constructive formal design methodology of automatic semantics-based program manipulation algorithms and tools (e.g. [7]).

Semantics have been studied in the framework of abstract interpretation [8,9] and compared according to their relative precision. A number of semantics including among others small-step, big-step, termination and nontermination semantics, Plotkin's natural, Smyth's demoniac, Hoare's angelic relational and corresponding denotational semantics, Dijkstra's weakest precondition and weakest liberal precondition predicate transformers and Hoare's partial and total axiomatic semantics have all been derived by successive abstractions starting from an operational maximal trace semantics of a transition system. This results in a hierarchy of semantics providing a complete account of the structure and relative precision of most well-known semantics of programming languages [10].

Program transformation (such as online and offline partial evaluation, program monitoring (e.g. for security policy enforcement or scheduling), etc.) is an abstract interpretation [11] where the program syntactic transformation is ann effective approximation of a corresponding undecidable transformation of the program semantics. The correctness of this program transformation is expressed as an observational equivalence of the subject and transformed semantics at some level of abstraction.

D. Bošnački and S. Leue (Eds.): SPIN 2002, LNCS 2318, pp. 2–5, 2002.

Typing that is formal type systems and type inference algorithms, is an approximation of the denotational semantics of higher-order functional programs [12]. The abstraction is powerful enough to show statically that “typable cannot go wrong” in that the denotational semantics of these programs cannot raise at run-time those errors excluded by typing. This point of view leads to a hierarchy of type systems, which is part of the lattice of abstract interpretation of the untyped lambda-calculus. The hierarchy includes classical Milner/Mycroft and Damas/Milner polymorphic type schemes, Church/Curry monotypes and Hindley principal typing algorithm as well as new à la Church/Curry polytype systems.

Model-checking classical linear-time and branching -time state based algorithms are sound and complete abstract interpretations of the trace-based semantics of transition systems [13]. Surprisingly, for the $\overset{\curvearrowleft}{\mu}{}^{\star}$-calculus, a novel general temporal specification language featuring a natural and rich time-symmetric trace-based semantics, model-checking turned out to be incomplete, even for finite systems [13]. Moreover, any model-checking for the $\overset{\curvearrowleft}{\mu}{}^{\star}$-calculus abstracting away from sets of traces will be necessarily incomplete [14].

Static program analysis is the first and most prevalent application of abstract interpretation [1,3,4,5]. By effective approximation of the fixpoint semantics of programs through abstraction [4,5] and convergence acceleration [4,15], a program analyzer will produce maybe incomplete but always sound information about the run-time behavior of programs. Abstract interpretation provides a general theory behind all programs analyzers, which only differ in their choice of considered programming languages (e.g. imperative [16,17], parallel [18,19], functional [20], logic [21], etc), program properties (among many others, run-time errors [16,22], precision [23], security [24,25], fair liveness [26], probabilistic termination [27], etc) and their abstractions. Finally, we will discuss the various possible designs of program analyzers, from general-purpose to application-specific ones.

References

1. Cousot, P.: Méthodes itératives de construction et d’approximation de points fixes d’opérateurs monotones sur un treillis, analyse sémantique de programmes. Thèse d’État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, FR (1978)
2. Cousot, P.: Semantic foundations of program analysis. In Muchnick, S., Jones, N., eds.: Program Flow Analysis: Theory and Applications. Prentice-Hall (1981) 303–342
3. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: 4^{th} POPL, Los Angeles, CA, ACM Press (1977) 238–252
4. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: 6^{th} POPL, San Antonio, TX, ACM Press (1979) 269–282
5. Cousot, P., Cousot, R.: Abstract interpretation frameworks. J. Logic and Comp. **2** (1992) 511–547

6. Cousot, P.: Abstract interpretation based formal methods and future challenges, invited paper. In Wilhelm, R., ed.: « Informatics — 10 Years Back, 10 Years Ahead ». Volume 2000 of LNCS. Springer-Verlag (2000) 138–156
7. Cousot, P.: The calculational design of a generic abstract interpreter. In Broy, M., Steinbrüggen, R., eds.: Calculational System Design. Volume 173. NATO Science Series, Series F: Computer and Systems Sciences. IOS Press (1999) 421–505
8. Cousot, P.: Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. ENTCS **6** (1997) `http://www.elsevier.nl/locate/entcs/volume6.html`, 25 pages.
9. Cousot, P., Cousot, R.: Inductive definitions, semantics and abstract interpretation. In: 19^{th} POPL, Albuquerque, NM, ACM Press (1992) 83–94
10. Cousot, P.: Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. Theoret. Comput. Sci. (2002) To appear (Preliminary version in [8]).
11. Cousot, P., Cousot, R.: Systematic design of program tansformation frameworks. In: 29^{th} POPL, Portland, OR, ACM Press (2002) 178–190
12. Cousot, P.: Types as abstract interpretations, invited paper. In: 24^{th} POPL, Paris, FR, ACM Press (1997) 316–331
13. Cousot, P., Cousot, R.: Temporal abstract interpretation. In: 27^{th} POPL, Boston, MA, ACM Press (2000) 12–25
14. Ranzato, F.: On the completeness of model checking. In Sands, D., ed.: Proc. 10^{th} ESOP '2001. Genova, IT, 2–6 Apr. 2001, LNCS 2028, Springer-Verlag (2001) 137–154
15. Cousot, P., Cousot, R.: Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In Bruynooghe, M., Wirsing, M., eds.: Proc. 4^{th} Int. Symp. PLILP '92. Leuven, BE, 26–28 Aug. 1992, LNCS 631, Springer-Verlag (1992) 269–295
16. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proc. 2^{nd} Int. Symp. on Programming, Dunod (1976) 106–130
17. Cousot, P., Cousot, R.: Static determination of dynamic properties of recursive procedures. In Neuhold, E., ed.: IFIP Conf. on Formal Description of Programming Concepts, St-Andrews, N.B., CA, North-Holland (1977) 237–277
18. Cousot, P., Cousot, R.: Semantic analysis of communicating sequential processes. In de Bakker, J., van Leeuwen, J., eds.: 7^{th} ICALP. LNCS 85, Springer-Verlag (1980) 119–133
19. Cousot, P., Cousot, R.: Invariance proof methods and analysis techniques for parallel programs. In Biermann, A., Guiho, G., Kodratoff, Y., eds.: Automatic Program Construction Techniques. Macmillan (1984) 243–271
20. Cousot, P., Cousot, R.: Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages), invited paper. In: Proc. 1994 ICCL, Toulouse, FR, IEEE Comp. Soc. Press (1994) 95–112
21. Cousot, P., Cousot, R.: Abstract interpretation and application to logic programs. J. Logic Programming **13** (1992) 103–179 (The editor of J. Logic Programming has mistakenly published the unreadable galley proof. For a correct version of this paper, see `http://www.di.ens.fr/~cousot`.).
22. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In Danvy, ., Filinski, A., eds.: Proc. 2^{nd} Symp. PADO '2001. Århus, DK, 21–23 May 2001, LNCS 2053, Springer-Verlag (2001) 155–172

23. Goubault, É., Martel, M., Putot, S.: Asserting the precision of floating-point computations: a simple abstract interrpeter. In Le Métayer, D., ed.: Proc. 11^{th} ESOP '2002. Grenoble, FR, LNCS ???? Springer-Verlag (2002) ???–???
24. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: 14^{th} IEEE Computer security Foundations Workshop (CSFW-14), Cape Breton, CA, IEEE Comp. Soc. Press (2001) 82–96
25. Feret, J.: Abstract interpretation-based static analysis of mobile ambients. In Cousot, P., ed.: Proc. 8^{th} Int. Symp. SAS '01. Paris, FR, LNCS 2126, Springer-Verlag (2001) 413–431
26. Mauborgne, L.: Tree schemata and fair termination. In Palsberg, J., ed.: Proc. 7^{th} Int. Symp. SAS '2000. Santa Barbara, CA, US, LNCS 1824. Springer-Verlag (29 June – 1 Jul. 2000) 302–321
27. Monniaux, D.: An abstract analysis of the probabilistic termination of programs. In Cousot, P., ed.: Proc. 8^{th} Int. Symp. SAS '01. Paris, FR, LNCS 2126, Springer-Verlag (2001) 111–127

In *Proceedings of the Ninth International SPIN Workshop on Model Checking Software*, D. Bošnački and S. Leue (Eds.), Grenoble, France, April 11—12, 2002. Lecture Notes in Computer Science 2318, © Springer, Berlin, 2002, pp. 2–5.