Program Analysis: The Abstract Interpretation Perspective

Patrick Cousot
École Normale Supérieure, Paris

Program analysis should evolve from a disparate collection of methods and algorithms to a mature discipline founded on a well-established methodology. Abstract interpretation can be used as a basis for such a methodology.

1. PROGRAM ANALYSIS

Program semantic analysis consists of designing and writing program analyzers. A program analyzer is a program which takes as input data a (possibly annotated) program (written in some language) and (fully automatically) produces as output answers to questions about runtime properties which are valid for all (or sometimes for some) possible executions of that program. Because of undecidability or complexity, these answers are necessarily partial, but should be irrefutable.

Program analysis is both a theoretical activity (close to formal specification methods, semantics, etc.) and a practical activity (close to compiler design and development). In a sense this situation is quite creative: looking at a real and vast application problem is a fruitful guide to the process of formalization of the semantics of programming languages, proof methods, etc. This is also a difficult position akin to the old debate between pure and applied mathematics. Often practitioners are not worried about theories while theorists rarely take time to find applications of their theoretical work which is loosely coupled with practical problems.

2. DESIGN METHODOLOGY

Semantic analysis (of programs, systems, etc.) should become a mature discipline to be included in educational curricula in computer science. This requires the discipline to evolve from an ill-assorted collection of methods and algorithms to a large scope formal reasoning and design methodology. In particular, program analysis methods should be explained in language, program encoding, semantics, property, property encoding, and approximation independent way, to be of very broad scope and wide usefulness. This might be possible by reasoning on the approximation of the structures involved in semantical specifications. This effort would culminate in a general theory of semantical approximation leading to a tractable composable parameterized semantical analysis design methodology.

3. ABSTRACT INTERPRETATION

Abstract interpretation understood as a theory of semantic approximation is a basis for such a methodology. It relies on the idea that the specification of an analyzer is an approximation of a semantics, where concrete or exact properties are replaced by abstract or approximate properties. This idea of approximation, and dually of refinement, is central to computer science although it is often left informal. To be of general scope, its presentation should emphasize underlying concepts and therefore be presented in a language, semantics and property independent way thus providing modular techniques and tools which can be composed to design actual program analyzers which are correct (and efficient), by construction.

4. SEPARATION OF CONCERNS

The separation between:

—The specification of the program properties to be determined (e.g. by a semantics and an abstraction function);

—The presentation of the specification (using fixpoints, constraints, inference system, etc. [5]);

—The form of questions for interfacing the analyzer (complete or partial (demand-driven) abstract properties, etc.)

—The abstract semantic domains/algebra which are used in the analysis;

—The resolution algorithms (iterative fixpoint computation, direct resolution by elimination, etc.)

leads to clearly separated problems which can be studied independently. We discuss briefly below the specification of the program properties through a hierarchy of semantics and then the abstract semantic algebra which are used for program analysis.

5. HIERARCHIES OF SEMANTICS

Founding a program analysis methodology on a specific semantics (small/big step operational, denotational [14], weakest-preconditions, etc.) turns out not to be very convenient. This is because a program analysis methodology based on a compositional semantics (e.g. denotational or axiomatic) can hardly be formulated independently of a specific language syntax and a specific class of program properties. Consequently the choice of a specific semantics may lead to elegant or tortuous specifications of properties whether they are directly expressible by the semantics or not: think of invariance properties expressed in terms of a denotational semantics or termination in terms of a small-step operational semantics. Consequently no particular semantics can be considered as general purpose.

We suggest instead to consider hierarchies of semantics which can describe program properties, that is program executions at various levels of abstraction or refinement [2]. Then for a given class of properties there should be a natural choice of semantics in the hierarchy (for example strictness analysis is awkward to explain using a denotational semantics [18] but very simple using a *relational semantics* [3] since there is a simple way to explain how a relation can be understood as a logical approximation of another, whereas this is not so simple for functions. Relations are too abstract to express absence, or more generally comportment which requires a further refinement [4]).

6. HIERARCHIES OF ABSTRACT ALGEBRAS

To a unique encoding of abstract properties (e.g. using logical formulæ or term algebra), we prefer the idea of hierarchies of application independent abstract algebras (the term abstract algebra is better than abstract domain since the design must include abstract properties but also abstract operations, widenings, ...). An example of multi-uses abstract algebra is linear inequalities for the convex-hull approximation of a set of vectors of numbers by a polyhedron [7] which have been used to estimate the size of arguments of logic programs [1; 9] and for infinite state model-checking [12; 8] including hybrid systems [13]. Another example is BDDs originating from model-checking but also useful to cope with size explosion in strictness analysis [16]. Not enough such analysis independent abstract algebras are presently available, in particular for discrete structures (see a.o. [6; 10; 19]) as opposed to numerical structures where the spectrum is wider [7; 11; 15]. Some abstract algebras are used without being clearly identified (e.g. for type inference, see however [17] for references). In this area, mainly of algorithmic nature, not enough constructive methods and algorithms from algebra, functional analysis or discrete mathematics have been investigated. Such algorithmic problems are rather involved and require experts, but their results could be easily incorporated in portable widely distributed libraries as is the case e.g. in the fields of numerical analysis and operational research.

7. DESIGN OF PROGRAM ANALYZERS

From a more practical point of view, we think that the uniform design of:

— hierarchies of semantics,

— hierarchies of abstract algebras

at various levels of abstraction/refinement would lead to generic program analysis tools most of which could be included in reusable and composable libraries.

An ultimate practical goal would be to automate the generation of such semantic analysis tools primarily for program analysis and more generally for system specification analysis.

REFERENCES

[1] Cousot, P. and Cousot, R. 1992a. Abstract interpretation and application to logic programs. *Journal of Logic Programming 13*, 2–3, 103–179.

[2] Cousot, P. and Cousot, R. 1992b. Inductive definitions, semantics and abstract interpretation. In *Conf. Rec. Ninthteenth ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages* (Albuquerque, New Mexico, 1992), pp. 83–94. ACM Press.

[3] Cousot, P. and Cousot, R. 1993. Galois connection based abstract interpretations for strictness analysis, invited paper. In D. Bjø rner, M. Broy, and I. Pottosin Eds., *Proc. Int. Conf. on Formal Methods in Programming and their Applications*, Academgorodok, Novosibirsk, Russia, Lecture Notes in Computer Science 735, pp. 98–127. Springer-Verlag.

[4] Cousot, P. and Cousot, R. 1994. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages), invited paper. In *Proc. 1994 Int. Conf. on Computer Languages*, Toulouse, France (16–19 May 1994), pp. 95–112. IEEE Computer Society Press.

[5] Cousot, P. and Cousot, R. 1995a. Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic form,

invited paper. In P. Wolper Ed., *Proc. Seventh Int. Conf. on Computer Aided Verification, CAV '95*, Liège, Belgium, Lecture Notes in Computer Science 939 (3–5 July 1995), pp. 293–308. Springer-Verlag.

[6] Cousot, P. and Cousot, R. 1995b. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proc. Seventh ACM Conf. on Functional Programming Languages and Computer Architecture* (La Jolla, Ca., 25–28 June 1995), pp. 170–181. ACM Press.

[7] Cousot, P. and Halbwachs, N. 1978. Automatic discovery of linear restraints among variables of a program. In *Conf. Rec. Fifth ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages* (Tucson, Arizona, 1978), pp. 84–97. ACM Press.

[8] Cridlig, R. 1995. Semantic analysis of shared-memory concurrent languages using abstract model-checking. In *Proc. ACM Symp. on Partial Evaluation and Semantics-Based Program Manipulation, PEPM '95*, La Jolla, Ca., 21–23 June 1995 (June 1995). ACM Press.

[9] Debray, S. 1994. Formal bases for dataflow analysis of logic programs. In G. Levi Ed., *Advances in Logic Programming Theory*, International Schools for Computer Scientists, Section 3, pp. 115–182. Clarendon Press, Oxford, UK.

[10] Deutsch, A. 1992. A storeless model of aliasing and its abstraction using finite representations of right-regular equivalence relations. In *Proc. 1992 Int. Conf. on Computer Languages*, Oakland, California (20–23 April 1992), pp. 2–13. IEEE Computer Society Press.

[11] Granger, P. 1991. Static analysis of linear congruence equalities among variables of a program. In S. Abramsky and T. Maibaum Eds., *Proc. of the Int. Joint Conf. on Theory and Practice of Software Development, TAPSOFT '91*, Brighton, U.K., Volume 1 (CAAP '91), Lecture Notes in Computer Science 493, pp. 169–192. Springer-Verlag.

[12] Halbwachs, N. 1994. About synchronous programming and abstract interpretation. In B. Le Charlier Ed., *Proc. of the Static Analysis Symp., SAS '94*, Namur Belgium, 20–22 Sept. 1994, Lecture Notes in Computer Science 864, pp. 179–192. Springer-Verlag.

[13] Halbwachs, N., Proy, J.-É., and Raymond, P. 1994. Verification of linear hybrid systems by means of convex approximations. In B. Le Charlier Ed., *Proc. of the Static Analysis Symp., SAS '94*, Namur Belgium, 20–22 Sept. 1994, Lecture Notes in Computer Science 864, pp. 223–237. Springer-Verlag.

[14] Jones, N. and Nielson, F. 1995. Abstract interpretation: a semantic-based tool for program analysis. In S. Abramsky, G. D.M., and M. T.S.E. Eds., *Semantic modelling*, Number 4 in Handbook of Logic in Comp. Sci. Clarendon Press.

[15] Masdupuy, F. 1992. Array operations abstraction using semantic analysis of trapezoid congruences. In *Proc. ACM Int. Conf. on Supercomputing, ICS '92* (Washington D.C., July 1992), pp. 226–235.

[16] Mauborgne, L. 1994. Abstract interpretation using TDGs. In B. Le Charlier Ed., *Proc. of the Static Analysis Symp., SAS '94*, Namur Belgium, 20–22 Sept. 1994, Lecture Notes in Computer Science 864, pp. 363–379. Springer-Verlag.

[17] Monsuez, B. 1995. System F and abstract interpretation. In A. Mycroft Ed., *Proc. of the Static Analysis Symp., SAS '95*, Glasgow, UK, 25–27 Sept. 1995, Lecture Notes in Computer Science 983, pp. 279–295. Springer-Verlag.

[18] Mycroft, A. 1981. Abstract interpretation and optimising transformations for applicative programs. Ph. D. thesis, Department of Computer Science, University of Edinburgh, Scotland.

[19] Venet, A. 1996. Abstract cofibred domains: Application to the alias analysis of untyped programs. In R. Cousot and D. Schmidt Eds., *Proc. of the Static Analysis Symp., SAS '96*, Aachen, Germany, 24–26 Sept. 1996, Lecture Notes in Computer Science 1145, pp. 368–382. Springer-Verlag.