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 ASTREÉ 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 ASTREÉ 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 acceptance 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 parcelized 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].

**References**


