Abstract Interpretations
a Critical Survey of Selected Papers
written by Patrick Cousot and Radhia Cousot

Mátyás Sustik

May 2, 2003

Abstract

I survey the abstract interpretation framework as presented in selected papers of Patrick Cousot and Radhia Cousot. The mathematical background and the application of the method is explored, evaluated.

1 Introduction and Motivation

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 behavior of a computer system in interaction with any possible environment. By approximation of this semantics we mean the use of an abstraction which ignores some irrelevant detail and henceforth makes it possible to solve the verification problem. Conservative means that the approximation can never lead to an erroneous conclusion, but may allow ‘false positives’. [8].

The increase in hardware capabilities lead to an explosion in the size of the programs as well. In order to ensure the reliability of today’s systems, tools need to be developed that analyze statically the run-time behavior (correctness) of programs. Since the correctness problem is undecidable in general the tools used are all partial or incomplete and the best we can hope for is that they solve with sufficient efficiency important classes of problems arising in practice.

The mechanical verification tools are all quite similar and essentially differ in the choice regarding what approximations are used and how automated their construction in relation to the specific problem. The stated purpose of abstract interpretation framework is to formalize these approximations. [5]. In more direct terms: abstractions are used in conjunction with program static analysis, model checking and theorem proving tools to cope with infinite state spaces and/or with computer resource limitations. The abstract interpretation framework is a formal

1 Also referred to as authors in this survey
basis for making the abstractions explicit and henceforth making it possible to reason about their soundness and (in)completeness.

In this survey I concentrate on abstract interpretations used in software verification [2] [5] [9] [10].

2 Abstract Interpretation

Abstract interpretation theory is the theory of abstraction of structures. For example program verification deals with properties (and with the sets of objects having these properties) and so abstract interpretation can be formulated in an application independent setting as a theory for approximating sets and set operations (corresponding to properties) as considered in set (and category) theory, including inductive definitions [5].

A more restricted understanding of abstract interpretations is to think of it as describing the behavior of dynamic discrete systems. Since such behaviors can be characterized by fixpoints, an essential part of the theory is aimed to provide constructive and effective methods for fixpoint approximation and checking by abstraction [5].

The abstract interpretation framework grow out of (static) program analysis [1]. The early definition is essentially based on the use of Galois connections however the term appears only in later publications only (for example [2]). The Galois connection is used to establish the relation between the abstract and concrete domains.

The concept of a domain is used in many different meanings and it is unfortunate that the authors were not always specific when they used this term in their various papers. In the first development the domains are lattices, however later they also discuss the use weaker notions such as semi-lattices and partially ordered sets. The domains consist of properties describing the behavior of the system according to various semantics such as trace, operational, denotational.

The existence of the Galois connection means that there is a unique best approximation for every concrete element. In [2] the framework has been extended by relaxing the uniqueness requirement to allow the study of “more flexible but less elaborated variants of the original framework”. The authors also note in [2] that “theoretically it is always possible to satisfy this hypothesis” (that is the existence of unique best approximation) by extending the abstract domain; however this solution might be prohibited due to explosion in size or other practical constraints.

2.1 Galois connection

An abstract interpretation is described by a Galois connection between the $L$ concrete and $L^\exists$ abstract domains which are classically represented by complete lattices. The Galois connection is realized by the $\alpha, \gamma$ monotone mappings between the $L$ and $L^\exists$ domains:

$$\alpha : <L, \subseteq> \longrightarrow <L^\exists, \subseteq^\exists> \quad \gamma : <L^\exists, \subseteq^\exists> \longrightarrow <L, \subseteq>$$
such that $\forall x \in L : \forall y \in L^\delta : \alpha(x) \sqsubseteq^\delta y \iff x \sqsubseteq y \gamma(y)$. The intuition is that in the concrete domain any $x \in L$ can be approximated by an $x'$ such that $x \sqsubseteq x'$; that is $x'$ is a weaker property implied by $x$. Conversely in the abstract domain $L^\delta$, a $y$ can be an approximation to an $x$ provided $x \sqsubseteq y \gamma(y)$. The purpose of the $\alpha$ mapping is to specify the best or most precise approximation available (having the $L^\delta$ abstract domain and the $\gamma$ mapping fixed) [9].

The authors present a proof in [9] that the $\alpha$ mapping of a Galois connection is indeed captures the unique best approximation. To this end they claim and prove that $\alpha(x)$ is an approximation indeed and that for any $y \in L^\delta$ abstract approximation the corresponding concrete approximations satisfy: $\gamma(\alpha(x)) \sqsubseteq \gamma(y)$. It is not clear why this latter property captures a best approximation and why not $\alpha(x) \sqsubseteq \gamma(y)$ is used instead. For a monotone $\gamma$ (which is the case in a Galois connection) the two conditions are of course equivalent.

**Theorem** If $\alpha$ and $\gamma$ are mappings of a Galois connection between $< L, \sqsubseteq >$ and $< L^\delta, \sqsubseteq^\delta >$ as defined above and $\gamma$ is monotone then $\alpha(x)$ is a best approximation meaning that it is an approximation and that for any $y \in L^\delta$ approximation $\gamma(\alpha(x)) \sqsubseteq \gamma(y)$.

**Proof.** By reflexivity of $\sqsubseteq^\delta$ and the definition of the Galois connection we have $\alpha(x) \sqsubseteq^\delta \alpha(x)$ implying $x \sqsubseteq \gamma(\alpha(x))$ establishing that $\alpha(x)$ is an approximation indeed.

Now suppose that $y \in L^\delta$ is an abstract approximation of $x$ that is $x \sqsubseteq y \gamma(y)$ by definition. The Galois connection property implies that $\alpha(x) \sqsubseteq^\delta y$ and finally the monotonicity of $\gamma$ yields that $\gamma(\alpha(x)) \sqsubseteq \gamma(y)$ □.

The natural question arises whether a given $\gamma$ as above determines the $\alpha$ or not. The answer is yes if we assume that $\gamma$ is continuous (we work with complete lattices as in the classical framework):

**Theorem** If $\gamma : < L^\delta, \sqsubseteq^\delta > \rightarrow < L, \sqsubseteq >$ is a monotone continuous map between the $L^\delta$ and $L$ complete lattices then the below defined $\alpha$ and the given $\gamma$ form a Galois connection:

$$\alpha(x) = \inf\{y \mid x \sqsubseteq \gamma(y)\}.$$  

**Proof.** The use of infimum is proper since $L^\delta$ is a complete lattice. We need to prove that for any $y \in L^\delta$, $x \in L$ we have $\alpha(x) \sqsubseteq^\delta y \iff x \sqsubseteq \gamma(y)$. Suppose first that $x \sqsubseteq \gamma(y)$. The above definition included $y$ among the $L^\delta$ elements for which the infimum was taken therefore trivially: $\alpha(x) \sqsubseteq y$.

Secondly, suppose that $\alpha(x) \sqsubseteq^\delta y$. By the monotonicity of $\gamma$ we can deduce that $\gamma(\alpha(x)) \sqsubseteq \gamma(y)$. To finish the proof we only need to establish that $x \sqsubseteq \gamma(\alpha(x))$ to get $x \sqsubseteq \gamma(y)$ by transitivity of $\sqsubseteq$.

Since $\gamma$ is continuous $\gamma(\alpha(x))$ can be calculated as an infimum:

$$\gamma(\alpha(x)) = \inf\{\gamma(y) \mid x \sqsubseteq \gamma(y)\}.$$  

This formulation immediately yields that $x \sqsubseteq \gamma(\alpha(x))$ and we are done □.

The above theorem can be generalized noticing that we did not use all properties of complete lattices (for example the join operation was not mentioned). However generalizations might not be of much practical interest, because any partially ordered set can be embedded in a complete lattice preserving all upper and
lower bounds. This observation was made in relation to abstract interpretations in [1].

Finally to end this section about the basic mathematical properties and background of Galois connections we show that the monotonicity of the $\alpha$ and $\gamma$ mappings follows from the property prescribed by the Galois connection. This result is mentioned in the footnote on page 8 of [2].

**Theorem** If $\alpha : \langle L, \subseteq \rangle \rightarrow \langle L^\delta, \subseteq^\delta \rangle$ and $\gamma : \langle L^\delta, \subseteq^\delta \rangle \rightarrow \langle L, \subseteq \rangle$ are such mappings between the $\langle L, \subseteq \rangle$ and $\langle L^\delta, \subseteq^\delta \rangle$ complete lattices that for every $x \in L$ and $y \in L^\delta$:

$$\alpha(x) \subseteq^\delta y \iff x \subseteq \gamma(y)$$

then $\alpha, \gamma$ are monotone.

**Proof.** If $x_1 \subseteq x_2 \in L$ we can argue that $\alpha(x_2) \subseteq \alpha(x_2)$ implies $x_2 \subseteq^\delta \gamma(\alpha(x_2))$ and so $x_1 \subseteq \gamma(\alpha(x_2))$. Applying the Galois connection condition again yields: $\alpha(x_1) \subseteq \alpha(x_2)$. Interchanging $\alpha$ and $\gamma$ in the above argument gives the proof of the monotonicity of $\gamma \Box$.

### 2.2 Hierarchy of Abstractions

The abstractions and the various abstract domains of a given (fixed) domain can be arranged in a lattice where comparable domains are such that one can be acquired from the another using abstractions. The reason that a lattice structure exists is due to the fact that composition of abstractions is an abstraction (a simple consequence of the definition of Galois connection) and that in practice the final abstraction is obtained through several abstraction steps. In [5] the authors describe (the theory of) hierarchy of semantics and abstractions.

The different abstractions represent different approximations attempted to deal with the various verification problem(s) of a concrete system.

The authors explore different abstractions of a program execution. The finest (most precise) semantics they consider is trace semantics: a sequence of states observed at discrete time steps where the program transitions in an atomic step from state to state. An abstraction of this model can lead to the relational semantics where the traces are replaced with a pair consisting of the initial and final states for finite traces and a pair consisting of the initial state and a symbol marking non termination. A further abstraction which keeps only the second element of the pairs leads to the denotational semantics.

Further abstractions can be obtained by collecting all states appearing along some finite or infinite state. For example this method can yield the partial correctness semantics (non-termination ignored) and static, collecting semantics which target proving invariance properties of programs and can express properties about reachability.
A further example is the class of effective abstractions. An example is numerical abstractions where we can “forget” in a state all but properties/values of (say) two variables and the result can be a reduced state space with transitions between pairs of integers. Properties like bounds on the values of the variables can be expressed. The authors call non-relational abstractions the abstractions which ignore possible relationships between values. They list sign abstraction where only the sign of a variable is preserved and interval abstraction, congruence abstraction as further examples. In contrast among relational abstractions they list polyhedral abstraction which amounts to represent a set of values(points) with their convex hull and/or with the corresponding system of linear equations.

Symbolic abstractions refer to abstractions of complex data structure like pointers, trees, graphs, communication structures used in programs. The authors note in [5] that such abstractions are difficult to find and cite one example where binary decision diagrams and tree schemata is used to abstract sets of infinite trees.

2.3 Fixpoint Abstraction

A fixpoint of a function $F$ can often be obtained as the limit of the iterations of $F$ from a given initial value. This is the case for the $\mu$-calculus asserted by the Tarski-Knaster theorem.

The abstraction of the fixpoint can often be calculated by taking the limit of the abstracted iteration sequence. Continuity of the function and of the $\alpha$ abstraction mapping in the Galois connection ensures this property:

**Theorem** If $F : L \to L$ is a monotone continuous function on the $< L, \sqsubseteq >$ lattice so that its least fixed point $x$ can be calculated as the limit of a stationary transfinite sequence satisfying:

$$x_0 = \bot, \quad x_{\lambda + 1} = F(x_\lambda), \quad x_\lambda = \sup_{\beta < \lambda} x_\beta,$$

then the least fixed point of the corresponding abstract $\alpha(x_\lambda)$ sequence is $\alpha(x)$.

**Proof.** Immediate consequence of the continuity of $\alpha$ and the fact that $x_\lambda$ form a chain. □

In general (when $\alpha$ is not continuous) we cannot claim that the abstract fixpoint is the best approximation to the concrete fixpoint; it can be a weaker approximation [5].

2.4 Extended Abstract Interpretation Framework

In [2] the authors present a generalized theory of abstract interpretations, relaxing requirements on the domains and properties of the mappings. First of all they require the concrete and abstract domains to be a chain complete partially ordered set (cpo) only.

The new definition however does not ensure the soundness of the abstraction automatically; they introduce the so called soundness relation and various assumptions about the abstract approximation. Among these are for example the assumption of the existence of abstract approximation which was immediate in the Galois framework (and unicity of best approximation was also granted).
The authors present a method or proof skeleton for establishing the soundness condition. The method relies on the semantics being defined by fixed points which are calculated by an iteration and that iteration is lifted to the abstract level. Assumptions about the convergence of said sequences lead to soundness proofs.

2.5 Model checking in the Abstract Interpretation Framework

In model checking the abstractions are usually state to state abstractions given by an $\alpha : S \rightarrow S^\alpha$ function. The $\alpha$ mapping naturally extends to subsets of $S$ by the following definition: $\alpha(X) = \{\alpha(s) \mid s \in X\}$ Note that sets of states are often referred to as properties, and in this case we make these to form the domains of the Galois connection:

$$\alpha : <P(S), \subseteq> \rightarrow <P(S^\alpha), \subseteq> \quad \gamma : <P(S^\gamma), \subseteq> \rightarrow <P(S), \subseteq>,$$

where the extended $\gamma$ is defined on $P(S^\alpha)$ as follows: $\gamma(Y) = \{x \in S \mid \alpha(x) \in Y\}$. The authors state in [9] that this connection can be derived from the transition relation.

In a general discussion on reachability the authors introduce $\text{post}_{[\alpha]}$ called post-image which assigns to every set of states the set of states reachable from any state from the set. The $\text{pre}_{[\alpha]}$ operator is defined similarly denoting the pre-image. They claim that the above Galois connection is of this form as well.

It is not clear how the above $\alpha$ and $\gamma$ relates to the $\text{post}_{[\alpha]}$ and $\text{pre}_{[\alpha]}$ mappings. I assume that some definition has been omitted that is needed to complete the formulation of abstract model checking in the abstract interpretation framework.

The authors note also in [9] that the need to restrict to state to state abstractions stems from the fact how existing model checkers handle the abstract semantics. They also mention that since not all abstractions are state to state therefore these are beyond the scope of abstract model checking. Examples for this claim are interval abstraction and polyhedral abstraction.

3 Applications

In this section I overview applications of the abstract interpretation framework.

3.1 Fixpoint Checking

The authors in [6] discuss various fixpoint checking algorithms starting from the traditionally used Tarski-Knaster like fixpoint finding algorithms and then algorithms operating in presence of abstractions. They also combine two classical algorithms into a new one.

The abstraction algorithms can accommodate infinite state systems. The author shows that with the assumption of partially complete abstractions any partially complete abstract domain must contain the exact abstraction of an invariant as computed by the traditional algorithms.
The problem in fixpoint checking is to assert that the (least) fixpoint of some $F$ monotone $L \rightarrow L$ function is contained in some $S$ subset. Properties are often defined as fixed points of functions the prime examples are temporal logic and $\mu$-calculus.

Since the usual method of using iterations to calculate the fixed point can be non-terminating for infinite state systems (for example in static in program analysis) abstractions are used with the hope that the abstracted fixpoint is related to the concrete one in such a manner which allows the completion of the analysis.

The following algorithm is presented in [6] as the first fixpoint checking algorithm: $X := I; G_0 := (X \leq S);$ 
while $G_0$ do 
$X' := I \lor F(X);$ 
$G_0 := (X \neq X') \& (X' \leq S);$ 
$X := X'$; 
od; 
return $(X \leq S);$ 

The authors present both a classical and an abstract interpretation framework proof which asserts that if the algorithm terminates then it correctly decides whether the least fixed point of $F$ is contained in $S$ or not. Notice that the algorithm is essentially the same as the one used to calculate $AFp$ given a Kripke structure.

The adjoint of $F$ denoted by $\widetilde{F}$ is a monotone $L \rightarrow L$ function such that it forms a Galois connection with $F$. The adjoin does not necessarily exist, but we assume its existence for the rest of the discussion.

The second algorithm is a dual of the first: $Y := S; G_0 := (I \leq Y);$ 
while $G_0$ do 
$Y' := S \land \widetilde{F} (X);$ 
$G_0 := (Y \neq Y') \& (I \leq Y');$ 
$Y := Y'$; 
od; 
return $(I \leq Y);$ 

Again not the similarity to the algorithm used to calculate $AGp$ for a Kripke structure.

The third algorithm is a combination of the previous two as follows: $X := I; Y := S; G_0 := (X \leq Y);$ 
while $G_0$ do 
$X' = I \lor F(X)Y' := S \land \widetilde{F} (X);$ 
$G_0 := (X \neq X') \& (Y \neq Y') \& (X' \leq Y');$ 
$X := X'; Y := Y'$; 
od; 
return $(X \leq Y);$ 

The authors argue that this algorithm is more time-efficient than the previous one. The strength of this claim should be established through an implementation.

The abstract fixpoint checking algorithm operates in the presence of a Galois
connection realized by the $\alpha$ and $\gamma$ mappings:

\[
X := \alpha(I); Go := (\gamma(X) \leq S);
\]

\begin{verbatim}
while Go do
    X' := \alpha(I \lor F(\gamma(X))); 
    Go := (X \neq X' \& \& (\gamma(X') \leq S)); 
    X := X';
end
\end{verbatim}

\textbf{return} if $(\gamma(X) \leq S)$ then \textbf{true} else \textbf{I don't know};

The partial completeness of this algorithm is examined the authors establish that an abstraction is partially complete for the algorithm if and only if $\alpha(L)$ contains an abstract value $A$ such that $\gamma(A)$ is an invariant for $< F, I, S >$. And finally more abstract fixpoint checking algorithms are presented after introducing notions of a dual Galois connection.

### 3.2 Mycroft’s strictness analysis

By considering that non-terminating and erroneous behaviors are essentially equivalent, call-by-need can be replaced by call-by-value in functional programs provided the arguments are evaluated in the function body or in later (recursive) function calls. Alan Mycroft’s strictness analysis [13] is designed to recognize these situations so that the substitution (part of an optimization step) can be safely done.

In the paper [3] the authors re-derive using the abstract interpretation framework Mycroft’s strictness analysis and claim to improve the related algorithm by Johnsson by adding a subsequent dependence-free abstraction to Mycroft’s dependence sensitive method.

It is not convincing however that theory of abstract interpretations actually helps in the design of abstractions used in this context. There were no quantitative measurements presented to support the claim of an improved algorithm since no actual implementation was referenced.

### 3.3 Algebraic Polynomial systems

The application of abstract interpretation framework to algebraic polynomial systems is presented in [4], generalizing classical formal language theoretical results and context-free grammar flow-analysis algorithms. Results include extensions to infinite terms.

It had been known that context-free grammars can be related to algebraic polynomial systems. The authors formalize this connection in the abstract interpretation framework. Abstractions of algebraic polynomial systems are derived from abstractions of context-free grammars and in some cases this leads to generalizations for infinite terms.

It remains to be seen how the abstract interpretation framework will help to analyze semantics of languages and/or yield information on the program executions abstracted by algebraic polynomial systems.
3.4 Temporal Logic

The abstractions of temporal logic is explored in [7] in a general syntax, semantics and abstraction independent setting. The theory is applied to a generalization of the $\mu$-calculus with new reversal and abstraction modalities. The soundness and completeness of the abstractions are discussed and the sources of incompleteness is pointed out and consequently a relatively complete sub-logics is identified.

It is shown that model checking is an abstract interpretation of the trace based semantics of temporal logic. In case of abstract model checking an additional state based abstraction is added. The source of incompleteness of these abstractions are pointed out.

These results are yet to prove their usefulness in creating and/or verifying abstractions of a given system. The most promising aspect is the potentially better understanding of the reasons and extent for an abstraction being incomplete.

3.5 Reachability Analysis

In [9] the authors investigate abstractions in software verification illustrated on reachability analysis and abstract testing. They define/discuss exact formal methods, abstract interpretation based formal methods, automatic abstraction and abstraction refinement and precise fixpoint checking in presence of approximations.

3.6 Grammars and Parsing

The paper [11] shows that Earley’s parsing algorithm is an abstract interpretation of refinement of the derivation semantics of context-free grammars.

The parsing algorithm discussed is a least fixpoint calculating algorithm and this indicates that a formulation using notions related to partially ordered sets; hence using abstract interpretations is possible. However it is not clear what benefit if any this formulation can contribute to better understand the semantics of context-free grammars.

4 Conclusion and Evaluation

In ABC of Reading (Faber and Faber, London, 1979, pp. 39-40), Ezra Pound makes the following observations:

“When you start searching for ‘pure elements’ in literature you will find that literature has been created by the following classes of persons:

1. Inventors. Men who found a new process, or whose extant work give us the first known example of a process.
2. The masters. Men who combined a number of such processes, and who used them as well as or better than the inventors.
3. The diluters. Men who came after the first two kinds of writer, and couldn’t do the job quite as well.
4. Good writers without salient qualities. Men who are fortunate enough to be born when the literature of a given country is in good working order, or when some particular branch of writing is ‘healthy’. […]

5. Writers of belles-lettres. That is, men who didn’t really invent anything, but who specialized in some particular part of writing, who couldn’t be considered as ‘great men’ or as authors who were trying to give a complete presentation of life, or of their epoch.

6. The starters of crazes.”

Replace “literature” with “computer science research” in the above to get an only slightly exaggerated description of where we stand today. I argue that the abstract interpretation framework puts its authors in the third category.

The theory is soundly based on the theory of lattices and partial orders, utilizing results such as the Tarski-Knaster fixed point theorem and morphisms of various kinds. It generalizes existing results without really highlighting how this would significantly advance either practical implementations or theoretical research. Rederiving existing results in a new framework might bring benefit by connecting research areas; but it seems that twenty five years has not been enough to establish this about the abstract interpretation framework. Abstraction and properties of abstractions have been and are being used by many researchers and it is not apparent what were or would be the value added by this theory. In [12] (co-authored by the inventors of the abstract interpretation framework) the design and implementation of a special-purpose program analyzer is presented. However there is no detail about the abstractions used during this development to be described in the abstract interpretation framework, and how was it used to prove the soundness and/or completeness properties of these abstractions.

During this survey I came across papers written by the authors about (specific) static program analysis methods and those writings were a worthy read. Possible that with the abstract interpretation framework the authors try to capture the general properties of the abstraction process and this will bring important developments in the future. At the moment however it seems that the generalizations lead to not so strong results while relies and uses (over?) complicated notation that does not help the research community to fully embrace and build on the ideas expressed in this theory.
References


