Abstract Interpretation of Algebraic Polynomial Systems

Patrick COUSOT
LIENS – DMI
École Normale Supérieure
45 rue d'Ulm
75230 Paris cedex 05
France
cousot@dmi.ens.fr
http://www.dmi.ens.fr/~cousot

Radhia COUSOT
LIX
CNRS & École Polytechnique
91140 Palaiseau cedex
France
rcousot@lix.polytechnique.fr
http://lix.polytechnique.fr/~rcousot

Abstract Interpretation

Abstract interpretation provides:

- A theory of discrete approximation to establish correspondences between various semantics of programming languages;
- A methodology to design algorithms for the static analysis of the dynamic behavior of programs.

References

[1] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In emphConf. Record of the 4^{th} Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 238–252, Los Angeles, California, 1977. ACM Press.

[2] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the 6^{th} Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press.

Applications of Abstract Interpretation

Design of

- Semantics,
- Proof methods,
- Static analyzers :
  - data flow analyzers,
  - type systems,
  - abstract model-checkers,
  - abstract debuggers, ...

of programming languages and systems.

Undecidability

- All problems considered by abstract interpretation are undecidable;
- The only possible answers are therefore approximate:
  - The answers are safe/sound/correct/conservative,
  - The answers may be partial/incomplete.

Idea of Approximation

Since interesting program properties are undecidable, no automatic program semantic analysis method can be complete. Automation implies approximation:

- **Proof methods:** unable to prove the following theorem: …10 pages of unreadable formulæ…;
- **Model-checking:** 10 hours later: out of memory;
- **Debugging:** 10 years later: still no error found;
- **Abstract Interpretation:** true/$\top$ (i.e. I don't know).

Main Characteristics of Abstract Interpretation

- Automatic: no human intervention during the analysis (as opposed e.g. to *interactive proof methods*);
- Static/compile-time: without running the program for specific input data (as opposed e.g. to *profiling*);
- Safe/sound/correct/conservative: without omitting the effect of some runs (as opposed e.g. to *debugging*);
- Dynamic properties: semantic properties of the runtime behaviors (as opposed e.g. to *program metrology*);
- Infinite-state: no (finiteness) limitation on the cardinality of the set of states (as opposed e.g. to *model-checking*).

Principle of Abstract Interpretation

- Syntax;
- Standard semantics;
- Concrete properties;
- Collecting semantics;
- Abstract Properties;
- Abstraction/concretization;
- Abstract semantics.

The abstract semantics is a safe approximation of the collecting semantics so that all run-time behaviors (specified by the standard semantics) of programs (specified by the syntax) satisfy the abstract properties specified by the abstract semantics.

Syntax

- The syntax defines a set of valid programs;
- Polynomial systems:

$$
\begin{array}{rcll}
S & \to & ES \mid E & \text{polynomial system,} \\
E & \to & x = P & \text{equation,} \\
 & \mid & x = \Omega & \text{void equation,} \\
P & \to & M[d] + P \mid M[d] & \text{labeled polynomial,} \\
M & \to & x \mid f(M_1, \ldots, M_n) & \text{monomial } (n \geq 0).
\end{array}
$$

- Example of syntactically valid polynomial system:

$$
\begin{array}{rll}
A = & b(A, A) & [d_1] \\
 & + \; a & [d_2]
\end{array}
$$

Standard Semantics

- The standard semantics specifies the set of possible runtime behaviors of programs;
- An example of possible parallel derivation tree execution sequence for

$$\begin{array}{lll} A = & b(A, A) & [d_1] \\ & + \; a & [d_2] \end{array}$$

is

$$\begin{array}{ll} & \langle A \rangle \\ \Longrightarrow & A[d_1]b(\langle A \rangle, \langle A \rangle) \\ \Longrightarrow & A[d_1]b(A[d_1]b(\langle A \rangle, A), A[d_2]a) \\ \Longrightarrow & A[d_1]b(A[d_1](A[d_2]a, \langle A \rangle), A[d_2]a) \\ \Longrightarrow & A[d_1]b(A[d_1]b(A[d_2]a, A[d_2]a), A[d_2]a) \; . \end{array}$$

Standard Semantics (continued)

- The standard semantics of a polynomial system is (e.g.) the set of finite and infinite parallel derivation tree execution sequences for all variables;
- For a polynomial system $\mathcal{P}$, we define:
  - The semantic domain $\mathcal{D}[\![\mathcal{P}]\!]$: the set of derivation tree sequences for the signature of $\mathcal{P}$;
  - The standard semantics $\mathcal{S}[\![\mathcal{P}]\!] \in \wp(\mathcal{D}[\![\mathcal{P}]\!])$: the set of derivation tree execution sequences for $\mathcal{P}$.

Concrete Properties

- A concrete property of a program is a set of possible program standard semantics;
- The set of concrete properties of a polynomial system $\mathcal{P}$ is a complete boolean lattice:

$$\langle \wp(\wp(\mathcal{D}[\![\mathcal{P}]\!])), \subseteq, \emptyset, \wp(\mathcal{D}[\![\mathcal{P}]\!]), \cup, \cap, \neg \rangle$$

for subset inclusion $\subseteq$, that is logical implication.

Collecting Semantics

- A collecting semantics associates a concrete property (of a given class e.g. safety, liveness, . . .) to each program:

$$\mathcal{C}[\![\mathcal{P}]\!] \in \wp(\wp(\mathcal{D}[\![\mathcal{P}]\!]))$$

- The standard collecting semantics:

$$\underline{\mathcal{C}}[\![\mathcal{P}]\!] \stackrel{\text{def}}{=} \{\mathcal{S}[\![\mathcal{P}]\!]\}$$

is the strongest concrete property.

Abstract Properties

- The abstract properties correspond to a well-chosen and conveniently encoded subset of the concrete properties;
- The set of abstract properties is a complete lattice

$$\left\langle \mathcal{D}^\sharp[\![\mathcal{P}]\!], \leq, 0, 1, \vee, \wedge \right\rangle$$

for the approximation ordering $\leq$, corresponding to concrete subset inclusion/logical implication.

Abstraction/Concretization

- The correspondence between concrete and abstract properties is defined by a Galois connection [1]:

$$\langle \wp(\wp(\mathcal{D}[\![\mathcal{P}]\!])), \subseteq, \emptyset, \wp(\mathcal{D}[\![\mathcal{P}]\!]), \cup, \cap \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \left\langle \mathcal{D}^\sharp[\![\mathcal{P}]\!], \leq, 0, 1, \vee, \wedge \right\rangle$$

  - α: abstraction ($\alpha(P)$ is the best/strongest/most precise approximation of P in the abstract domain);
  - γ: concretization ($\gamma(Q)$ is the concrete meaning of the abstract property Q).

[1] For weaker models, see P. Cousot & R. Cousot. "Abstract interpretation frameworks". *J. Logic and Comp.*, 2(4):511–547, 1992.

Galois Connection

- By definition:

$$\langle \wp(\wp(\mathcal{D}[\![\mathcal{P}]\!])), \subseteq, \emptyset, \wp(\mathcal{D}[\![\mathcal{P}]\!]), \cup, \cap \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \left\langle \mathcal{D}^\sharp[\![\mathcal{P}]\!], \leq, 0, 1, \vee, \wedge \right\rangle$$

means:

$$\forall P : \forall Q : \alpha(P) \leq Q \iff P \subseteq \gamma(Q)$$

- If α is surjective, we write:

$$\langle \wp(\wp(\mathcal{D}[\![\mathcal{P}]\!])), \subseteq, \emptyset, \wp(\mathcal{D}[\![\mathcal{P}]\!]), \cup, \cap \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\rightarrow}} \left\langle \mathcal{D}^\sharp[\![\mathcal{P}]\!], \leq, 0, 1, \vee, \wedge \right\rangle$$

The Intuition Behind Galois Connections

- Functional approximation : all properties P have a most precise approximation $\gamma(\alpha(P))$;
- Loss of information: the approximation of property P is less precise than P;
- Monotonicity: if P is more precise than Q then the approximation of P is more precise than the approximation of Q;
- Idempotence: The approximation of the approximation of P is just the approximation of P.

Example of Abstraction for Polynomial Systems

I. Disjunctive Approximation

- A powerset is approximated by the elements in the subsets:

$$\alpha_1 \; : \; \wp(\wp(\mathcal{D}[\![\mathcal{P}]\!])) \mapsto \wp(\mathcal{D}[\![\mathcal{P}]\!])$$
$$\alpha_1(P) = \bigcup P$$

- Disjunctive properties are lost.

II. Safety Analysis

- A set of finite and infinite derivation tree sequences is approximated by the set of derivation trees found along these sequences:

$$\alpha_2(P) = \{d \mid \exists \sigma \in P : \exists \sigma', \sigma'' : \sigma = \sigma' d \sigma''\}$$

- Liveness properties (fairness, termination, ...) are lost.

III. Generated Terminal Language

- A set of derivation trees is approximated by the generated terminal language;
- For the polynomial system:

$$\begin{array}{rll} A = & b(A, A) & [d_1] \\ & + \; a & [d_2] \end{array}$$

we get:

$$\begin{array}{rcl} @(\langle A \rangle) & = & @(A) \\ @(A[d_1]b(X, Y)) & = & @(X)@(Y) \\ @(A[d_2]a) & = & a \\ \alpha_3(P) & = & \{@(t) \mid t \in P\} \end{array}$$

- Structural properties are lost. $\square$

Composing Abstractions

- The composition of Galois connections is a Galois connection;
- Example: for the powerset of finite and infinite parallel derivation tree execution sequences to the generated terminal language abstraction:

$$\langle \wp(\wp(\mathcal{D}[\![\mathcal{P}]\!])), \subseteq \rangle \; \underset{\alpha_3 \circ \alpha_2 \circ \alpha_1}{\overset{\gamma_1 \circ \gamma_2 \circ \gamma_3}{\leftrightarrows}} \; \langle \wp(\mathcal{T}^{\,2}), \subseteq \rangle$$

- The abstraction can be decomposed according to the structure of program semantics hence program properties;
- A great variety of abstractions has been designed to approximate the mathematical structures involved in defining the semantics of programming languages.

[2] $\mathcal{T}$ is the set of constants of the signature of the polynomial system

Abstract Semantics

- An abstract semantics associates an abstract property to each program $\mathcal{P}$:

$$\mathcal{S}^\sharp[\![\mathcal{P}]\!] \in \mathcal{D}^\sharp[\![\mathcal{P}]\!]$$

- The abstract semantics is a safe approximation of the collecting semantics:

$$\mathcal{C}[\![\mathcal{P}]\!] \subseteq \gamma(\mathcal{S}^\sharp[\![\mathcal{P}]\!])$$

The Abstract Interpretation Design Methodology

- Define the abstract semantics $\mathcal{S}^\sharp[\![\mathcal{P}]\!]$ by calculation, simplifying the expression $\alpha(\underline{\mathcal{C}}[\![\mathcal{P}]\!])$, using $\leq$-approximations for simplification purposes;
- The soundness

$$\mathcal{S}[\![\mathcal{P}]\!] \in \gamma(\mathcal{S}^\sharp[\![\mathcal{P}]\!])$$

of the abstract semantics is then by construction.

Denotational Generic Abstract Interpreters

- Abstract semantics can be presented in
  - denotational (fixpoint, compositional) style, by induction on the syntactic structure of programs;
  - generic style, by parameterization with basic abstract algebras;
- The compositional presentation is preserved by abstraction, hence can be used as a generic abstract interpreter.

Example: Generic Bottom-up Abstract Semantics of Polynomial Systems

- $\langle \mathcal{D}_s, \sqsubseteq, \bot, \sqcup \rangle$, polynomial system semantic domain (cpo);
- $\langle \mathcal{D}_p, \leq, \uplus \rangle$, polynomial semantic domain (poset);
- $\mathcal{S}[\![\mathcal{P}]\!] = \mathrm{lfp}^{\sqsubseteq} \mathcal{B}[\![\mathcal{P}]\!]$, fixpoint semantics where $\mathcal{B}[\![\mathcal{P}]\!]$ is monotonic and defined compositionally as:

$$\begin{array}{rcl@{\qquad}rcl}
\mathcal{B}[\![ES]\!]r &=& \mathcal{B}[\![E]\!]r \sqcup \mathcal{B}[\![S]\!]r & \mathcal{B}[\![x]\!]r &=& x\langle r\rangle \\
\mathcal{B}[\![x = P]\!]r &=& \langle 1\rangle \sqcup \langle x \circ\!\!\longrightarrow \mathcal{B}[\![P]\!]r\rangle r & \mathcal{B}[\![f(L)]\!]r &=& f\langle r\rangle(\mathcal{B}[\![L]\!]r) \\
\mathcal{B}[\![\Omega]\!]r &=& \langle \Omega\rangle & \mathcal{B}[\![M, L]\!]r &=& \mathcal{B}[\![M]\!]r \oplus \mathcal{B}[\![L]\!]r \\
\mathcal{B}[\![M + P]\!]r &=& \mathcal{B}[\![M]\!]r \uplus \mathcal{B}[\![P]\!]r & \mathcal{B}[\![c]\!]r &=& c\langle r\rangle
\end{array}$$

Generated Terminal Language Semantics

- $\langle \mathcal{X} \mapsto \wp(\mathcal{T}^\star), \dot{\subseteq}, \dot{\emptyset}, \dot{\cup} \rangle$, language generated by each variable $x \in \mathcal{X}$;
- $\langle \wp(\mathcal{T}^\star), \subseteq, \cup \rangle$, language generated by a polynomial;
- Basic abstract operations:

$\sqcup$	$\langle 1 \rangle x$	$\langle x \circ\!\!\longrightarrow L \rangle r y$	$\langle \Omega \rangle$	$\uplus$	$x\langle r \rangle$	$f\langle r \rangle(L)$	$\oplus$	$c\langle r \rangle$
$\dot{\cup}$ [3]	$\emptyset$	$(x = y \mathbin{?} L \mathbin{¿} \emptyset)$	$\emptyset$	$\cup$	$r(x)$	L	$\cdot$ [4]	$\{c\}$

- A straightforward generalization of Ginsburg & Rice and Schützenberger theorem on the fixpoint characterization of the language generated by a context free grammar.

Generated Terminal Language Semantics (bis)

- $\langle \mathcal{X} \mapsto \wp(\mathcal{T}^\star), \dot{\subseteq}, \dot{\emptyset}, \dot{\cup} \rangle$, language generated by each variable $x \in \mathcal{X}$ (complete lattice);
- $\langle \wp(\mathcal{T}^\star), \subseteq, \cup \rangle$, language generated by a polynomial (complete lattice);
- $\mathcal{S}[\![\mathcal{P}]\!] = \mathrm{lfp}^{\dot{\subseteq}}\ \mathcal{B}[\![\mathcal{P}]\!]$, language generated by polynomial system $\mathcal{P}$ where $\mathcal{B}[\![\mathcal{P}]\!]$ is $\dot{\subseteq}$-monotonic and defined compositionally as:

$$\begin{aligned} \mathcal{B}[\![ES]\!]r &= \mathcal{B}[\![E]\!]r \mathbin{\dot{\cup}} \mathcal{B}[\![S]\!]r \\ \mathcal{B}[\![x = P]\!]ry &= (x = y \mathbin{?} \mathcal{B}[\![P]\!]r \mathbin{¿} \emptyset) \\ \mathcal{B}[\![\Omega]\!]r &= \emptyset \\ \mathcal{B}[\![M + P]\!]r &= \mathcal{B}[\![M]\!]r \cup \mathcal{B}[\![P]\!]r \end{aligned} \qquad \begin{aligned} \mathcal{B}[\![x]\!]r &= r(x) \\ \mathcal{B}[\![f(L)]\!]r &= \mathcal{B}[\![L]\!]r \\ \mathcal{B}[\![M, L]\!]r &= \mathcal{B}[\![M]\!]r \cdot \mathcal{B}[\![L]\!]r \\ \mathcal{B}[\![c]\!]r &= \{c\} \end{aligned}$$

[3] $\dot{\cup}$: pointwise language join, for each variable $x \in \mathcal{X}$.
[4] $\cdot$: set of strings concatenation.

Composing Homomorphic/Approximate Abstractions

If

$$\langle \mathcal{D}_s, \sqsubseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{D}_s^\sharp, \sqsubseteq^\sharp \rangle \qquad \alpha(\langle x \circ\!\!\longrightarrow L \rangle r) =/\sqsubseteq^\sharp \langle x \circ\!\!\longrightarrow \alpha'(L) \rangle^\sharp \alpha(r)$$

$$\langle \mathcal{D}_p, \leq \rangle \underset{\alpha'}{\overset{\gamma'}{\leftrightarrows}} \langle \mathcal{D}_p^\sharp, \leq^\sharp \rangle \qquad \alpha'(L_1 \oplus L_2) =/\leq^\sharp \alpha'(L_1) \oplus^\sharp \alpha'(L_2)$$

$$\alpha(\langle \Omega \rangle) =/\sqsubseteq^\sharp \langle \Omega \rangle^\sharp \qquad \alpha'(f\langle L \rangle) =/\leq^\sharp f\langle \alpha'(L) \rangle^\sharp$$

$$\alpha(\langle 1 \rangle) =/\sqsubseteq^\sharp \langle 1 \rangle^\sharp \qquad \alpha'(c\langle L \rangle) =/\leq^\sharp c\langle \alpha'(L) \rangle^\sharp$$

then

$$\mathcal{S}^\sharp[\![\mathcal{P}]\!] = \mathrm{lfp}^{\sqsubseteq^\sharp}\ \mathcal{B}^\sharp[\![\mathcal{P}]\!] =/\sqsubseteq^\sharp \alpha(\mathrm{lfp}^{\sqsubseteq}\ \mathcal{B}[\![\mathcal{P}]\!]) = \alpha(\mathcal{S}[\![\mathcal{P}]\!]) .$$

12 bottom/up or top/down abstract semantics are given in the paper.

Difficulty of Abstract Interpretation

- The semantics of programming languages is complex;
- The task of designing and constructing a program analyzer is therefore also extremely complex (typically much more complex than a compiler);
- Very few specialists are available who are able to develop and maintain a static analyzer for realistic practical languages.

How can we help in the design and construction of program analyzers?

Tools for Constructing Static Analyzers

- Static analyzer generators (akin to compiler generator)?
- Use of intermediate languages:
  - (Typed) lambda-calculi: too expressive? too few general purpose abstractions?
  - Polynomial systems: many possible language-theoretic abstractions, not enough expressive?
- Use of general purpose abstract domains:
  - Polynomial systems can be used for a natural generalization of set based/grammar based program analysis [3].

References

[3] P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proc.* 7th *ACM Conference on Functional Programming Languages and Computer Architecture*, pages 170–181, La Jolla, California, 25–28 June 1995. ACM Press.