Challenges in Abstract Interpretation for Software Safety

Patrick Cousot
École normale supérieure, Paris, France

cousot@ens.fr www.di.ens.fr/~cousot

Franco-Japanese Workshop on Security
Keio University, Mita Campus, September 5–7, 2005

State of Practice in Software Engineering

An example among many others (Matlab code)

```matlab
get(gca,'children');
```

The software safety challenge for next 10 years

- Present-day software engineering is almost exclusively manual, with very few automated tools;
- Trust and confidence in specifications and software can no longer be entirely based on the development process (e.g. DO178B in aerospace software);
- In complement, quality assurance must be ensured by new design, modeling, checking, verification and certification tools based on the product itself.

Abstract Interpretation

Reference


Syntax of programs

\[ X \]
variables \( X \in X \)

\[ T \]
types \( T \in T \)

\[ E \]
arithmetic expressions \( E \in E \)

\[ B \]
boolean expressions \( B \in B \)

\[ D ::= T \ X ; \]

\[ | \quad T \ X ; D' \]

\[ C ::= X = E ; \]

\[ | \quad \text{while} \ B \ C' \]

\[ | \quad \text{if} \ B \ C' \ \text{else} \ C'' \]

\[ | \quad \{ \ C_1 \ldots \ C_n \ \}, (n \geq 0) \]

\[ P ::= D \ C \]

program \( P \in P \)

---

Values of given type:

\[ \mathcal{V}[T] : \text{values of type } T \in T \]

\[ \mathcal{V}[\text{int}] \overset{\text{def}}{=} \{ z \in \mathbb{Z} | \min_{\text{int}} \leq z \leq \max_{\text{int}} \} \]

Program states \( \Sigma[P] \):

\[ \Sigma[D \ C] \overset{\text{def}}{=} \Sigma[D] \]

\[ \Sigma[T \ X ;] \overset{\text{def}}{=} \{ X \} \mapsto \mathcal{V}[T] \]

\[ \Sigma[T \ X ; D] \overset{\text{def}}{=} (\{ X \} \mapsto \mathcal{V}[T]) \cup \Sigma[D] \]

---

Concrete Semantic Domain of Programs

Concrete semantic domain for reachability properties:

\[ \mathcal{D}[P] \overset{\text{def}}{=} \rho(\Sigma[P]) \]

sets of states

i.e. program properties where \( \subseteq \) is implication, \( \emptyset \) is false, \( \cup \) is disjunction.

---

1 States \( \rho \in \Sigma[P] \) of a program \( P \) map program variables \( X \) to their values \( \rho(X) \)
Concrete Reachability Semantics of Programs

\[ S[X = E; R] \overset{\text{def}}{=} \{ \rho[X \leftarrow E] | \rho \in R \cap \text{dom}(E) \} \]

\[ \rho[X \leftarrow v](X) \overset{\text{def}}{=} v, \quad \rho[X \leftarrow v](Y) \overset{\text{def}}{=} \rho(Y) \]

\[ S[\text{if } B \text{ } C'; R] \overset{\text{def}}{=} S[C'](B[R] \cup B[\neg B]R) \]

\[ B[R] \overset{\text{def}}{=} \{ \rho \in R \cap \text{dom}(B) | B \text{ holds in } \rho \} \]

\[ S[\text{if } B \text{ } C' \text{ else } C''; R] \overset{\text{def}}{=} S[C'](B[R] \cup S[C''](B[\neg B]R) \]

\[ S[\text{while } B \text{ } C'; R] \overset{\text{def}}{=} \text{let } W = \text{lfp}_\alpha \lambda X. R \cup S[C'](B[R]X) \text{ in } (B[\neg B]W) \]

\[ S[\{\}]; R] \overset{\text{def}}{=} R \]

\[ S[C_1 \ldots C_n]; R] \overset{\text{def}}{=} S[C_n] \circ \ldots \circ S[C_1] \quad n > 0 \]

\[ S[D \text{ } C; R] = S[C](\Sigma[D]) \quad \text{(uninitialized variables)} \]

Not computable (undecidability).

---

Reduced Product of Abstract Domains

To combine abstractions

\[ \langle \mathcal{D}, \subseteq \rangle \xrightarrow{\gamma_1} \langle \mathcal{D}^1, \subseteq_1 \rangle \text{ and } \langle \mathcal{D}, \subseteq \rangle \xrightarrow{\gamma_2} \langle \mathcal{D}^2, \subseteq_2 \rangle \]

the reduced product is

\[ \alpha(X) \overset{\text{def}}{=} \bigcap \{ (x, y) | X \subseteq \gamma_1(X) \land X \subseteq \gamma_2(X) \} \]

such that \( \subseteq \overset{\text{def}}{=} \subseteq_1 \times \subseteq_2 \) and

\[ \langle \mathcal{D}, \subseteq \rangle \xrightarrow{\gamma_1 \times \gamma_2} \alpha(\mathcal{D}), \subseteq \]

---

Example: \( x \in [1, 9] \land x \mod 2 = 0 \) reduces to \( x \in [2, 8] \land x \mod 2 = 0 \)

---

Approximate Fixpoint Abstraction

\[ F^* \overset{\gamma}{\rightarrow} F^\# \]

\[ F^\# \overset{\gamma}{\rightarrow} F^* \]

\[ F^* \overset{\gamma}{\rightarrow} F^\# \]

\[ F^\# \overset{\gamma}{\rightarrow} F^* \]

Not computable (undecidability).
Abstract Reachability Semantics of Programs

\[ S^\sharp[X = E;]R \defeq \alpha(\{\rho | X \leftarrow E[\rho] \mid \rho \in \gamma(R) \cap \mathrm{dom}(E)\}) \]

\[ S^\sharp[\text{if } B C']R \defeq S^\sharp[C'](B^\sharp[B]R) \cup B^\sharp[\neg B]R \]

\[ B^\sharp[B]R \defeq \alpha(\{\rho \in \gamma(R) \cap \mathrm{dom}(B) \mid B \text{ holds in } \rho\}) \]

\[ S^\sharp[\text{if } B C' \text{ else } C'']R \defeq S^\sharp[C']'(B^\sharp[B]R) \cup S^\sharp[C'']'(B^\sharp[\neg B]R) \]

\[ S^\sharp[\text{while } B C']R \defeq \mathop{\text{lfp}}\limits_{\lambda \mathcal{X}. R \cup S^\sharp[C']'(B^\sharp[B]\mathcal{X}) \text{ in } (B^\sharp[\neg B]\mathcal{W})} \]

\[ S^\sharp[\{\} ]R \defeq R \]

\[ S^\sharp[\{C_1 \ldots C_n\}]R \defeq S^\sharp[C_1] \circ \ldots \circ S^\sharp[C_n] \quad n > 0 \]

\[ S^\sharp[D C]R \defeq S^\sharp[C](\top) \quad (\text{uninitialized variables}) \]

---

Abstract Semantics with Convergence Acceleration

\[ S^\sharp[X = E;]R \defeq \alpha(\{\rho | X \leftarrow E[\rho] \mid \rho \in \gamma(R) \cap \mathrm{dom}(E)\}) \]

\[ S^\sharp[\text{if } B C']R \defeq S^\sharp[C']'(B^\sharp[B]R) \cup B^\sharp[\neg B]R \]

\[ B^\sharp[B]R \defeq \alpha(\{\rho \in \gamma(R) \cap \mathrm{dom}(B) \mid B \text{ holds in } \rho\}) \]

\[ S^\sharp[\text{if } B C' \text{ else } C'']R \defeq S^\sharp[C']'(B^\sharp[B]R) \cup S^\sharp[C'']'(B^\sharp[\neg B]R) \]

\[ S^\sharp[\text{while } B C']R \defeq \mathop{\text{lfp}}\limits_{\lambda \mathcal{X}. R \cup S^\sharp[C']'(B^\sharp[B]\mathcal{X}) \text{ in } (B^\sharp[\neg B]\mathcal{W})} \]

\[ S^\sharp[\{\} ]R \defeq R \]

\[ S^\sharp[\{C_1 \ldots C_n\}]R \defeq S^\sharp[C_1] \circ \ldots \circ S^\sharp[C_n] \quad n > 0 \]

\[ S^\sharp[D C]R \defeq S^\sharp[C](\top) \quad (\text{uninitialized variables}) \]

---

Convergence Acceleration with Widening

Applications of Abstract Interpretation

---

\[ \text{Note: } \mathcal{F} \text{ not monotonic!} \]
Applications of Abstract Interpretation (Cont’d)

- **Static Program Analysis** [POPL ’77], [POPL ’78], [POPL ’79]
  including **Dataflow Analysis** [POPL ’79], [POPL ’00], **Set-based Analysis** [FPCA ’95], **Predicate Abstraction** [Manna’s festschrift ’03], . . .

- **Syntax Analysis** [TCS 290(1) 2002]

- **Hierarchies of Semantics (including Proofs)** [POPL ’92], [TCS 277(1–2) 2002]

- **Typing & Type Inference** [POPL ’97]

All these techniques involve **sound approximations** that can be formalized by **abstract interpretation**

---

Reference

Static analysis tools

- Determine automatically from the program text program properties of a certain class that do hold at runtime (e.g. absence of runtime error);
- Based on the automatic computation of machine representable abstractions\(^3\) of all possible executions of the program in any possible environment;
- Scales up to hundreds of thousands lines;
- Undecidable whence false alarms are possible\(^4\)

Degree of specialization

- Specialization for a class of runtime properties (e.g. absence of runtime errors)
- Specialization for a programming language (e.g. PolySpace Suite for Ada, C or C++)
- Specialization for a programming style (e.g. C Global Surveyor)
- Specialization for an application type (e.g. ASTRÉE for embedded real-time synchronous\(^5\) autocodes)
  ⇒ The more specialized, the less false alarms\(^6\)!

ASTRÉE static analyzer (www.astree.ens.fr)
- ASTRÉE is a static program analyzer aiming at proving the absence of Run Time Errors (started Nov. 2001)
- C programs, no dynamic memory allocation and recursion
- Encompass many (automatically generated) synchronous, time-triggered, real-time, safety critical, embedded software
- Automotive, energy and aerospace applications
  ⇒ e.g. No false alarm on the electric flight control codes for the A340 (Nov. 2003) and A380 (Nov. 2004) generated from SAO/SCADE.

Ellipsoid Abstract Domain for Filters

2\(^{\text{nd}}\) Order Digital Filter:

\[
X_n = \begin{cases} 
\alpha X_{n-1} + \beta X_{n-2} + Y_n \\
I_n
\end{cases}
\]

- Computes \(X_n\)
  - The concrete computation is bounded, which must be proved in the abstract.
  - There is no stable interval or octagon.
  - The simplest stable surface is an ellipsoid.

\(^3\) sound but (in general) incomplete approximations.
\(^4\) cases when a question on the program runtime behavior cannot be answered automatically for sure
\(^5\) deterministic
\(^6\) but the less specialized, the larger commercial market (and the less client satisfaction)!
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;

BOOLEAN INIT;
float P, X;

void filter () {
  static float E[2], S[2];
  if (INIT) {
    S[0] = X; P = X; E[0] = X;
  }
  else {
    P = (((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4))
                  + (S[0] * 1.5)) - (S[1] * 0.7));
  }
  /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}

void main () {
  X = 0.2 * X + 5; INIT = TRUE;
  while (1) {
    X = 0.9 * X + 35; /* simulated filter input */
    filter (); INIT = FALSE;
  }
}

-- Reference

see http://www.astree.ens.fr/

---

Arithmetic-geometric progressions

Abstract domain: \((\mathbb{R}^+)^5\)

Concretization (any function bounded by the arithmetic-geometric progression):

\[
\gamma \in (\mathbb{R}^+)^5 \mapsto \varphi(\mathbb{N} \mapsto \mathbb{R})
\]

\[
\gamma(M, a, b, a', b') = 
\{ f \mid \forall k \in \mathbb{N} : |f(k)| \leq (\lambda x. ax + b \circ (\lambda x. ax' + b')^k)(M) \}
\]

-- Reference

see http://www.astree.ens.fr/

---

Potential overflow!

void main() {
  R = 0;
  while (TRUE) {
    __ASTREE_log_vars((R));
    if (I) { R = R + 1; }
    else { R = 0; }
    T = (R >= 100);
    __ASTREE_wait_for_clock();
  }
}

void main() {
  FIRST = TRUE;
  while (TRUE) {
    dev();
    FIRST = FALSE;
    __ASTREE_wait_for_clock();
  }
}

void main()

\[
\begin{align*}
|P| &\leq (15. + 5.87747175411e-39/1.19209290217e-07)^{\text{clock}}-5.87747175411e-39/1.19209290217e-07
\leq 23.0393526881
\end{align*}
\]
Towards System Verification Tools

Reference


Computer controlled systems

Software test

Abstractions: program → none, system → precise

– Very expensive
– Not exhaustive
– Extended during flight test period
– Late discovery of errors can delay the program by months (the whole software development process must be rechecked)
Software analysis & verification with ASTRÉE

Abstractions: program \(\rightarrow\) precise, system \(\rightarrow\) coarse

System analysis & verification by control engineers

Abstractions: program \(\rightarrow\) imprecise, system \(\rightarrow\) precise

- Exhaustive
- Can be made precise by specialization\(^8\) to get no false alarm
- No specification of the controlled system (but for ranges of values of a few sensors)
- Impossible to prove essential properties of the controlled system (e.g. controllability, robustness, stability)

\(^8\) To specific families of properties and programs

- The controller model is a rough abstraction of the control program:
  - Continuous, not discrete
  - Limited to control laws
  - Does not take into account fault-tolerance to failures and computer-related system dependability.
- In theory, SDP-based search of system invariants (Lyapunov-like functions) can be used to prove reachability and inevitability properties
- Problems to scale up (e.g. over long periods of time)
- In practice, the system/controller model is explored by discrete simulations (testing)
Exploring new avenues in static analysis

- Exhaustive (contrary to current simulations)
- Traditional abstractions (e.g. polyhedral abstraction with widening) seem to be too imprecise
- Currently exploring new abstractions (issued from control theory like ellipsoidal calculus using SDP)
- Prototype implementation in construction!

Abstractions: program $\rightarrow$ precise, system $\rightarrow$ precise
– Example of invariant translation: ellipsoidal → polyhedral

– The static analysis is easier on the system/controller model using continuous optimization methods

– The translated invariants can be checked for the system simulator/control program (easier than invariant discovery)

– Should scale up since these complex invariants are relevant to a small part of the control program only

\[ \text{Example of invariant translation: ellipsoidal} \rightarrow \text{polyhedral} \]\n
The invariant hypotheses on the controlled system are assumed to be true

– It remains to perform the control program analysis under these hypothesis

– The results can then be checked on the whole system (as in case 2, but now using refined invariants on the control program!)

– Iterating this process leads to static analysis by refinement of specifications

**Conclusion**

Abstractions: program → precise, system → precise

---

\[9\] For which floating point computations can be taken into account
Scientific and technologic objective

To develop formal tools to answer questions about software:
- from control model design to software implementation,
- for a wide range of design and software properties,
which would be general enough to benefit all software-intensive industries, and can be adapted to specific application domains.

References

[3] www.astree.ens.fr [5, 6, 7, 8, 9, 10, 11, 12]


THE END, THANK YOU

More references at URL www.di.ens.fr/~cousot
www.astree.ens.fr.


—— 53 ——

