Abstract Interpretation–based Formal Verification of Complex Computer Systems

Patrick Cousot
Jerome C. Hunsaker Visiting Professor
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
cousot@mit.edu  www.mit.edu/~cousot

École normale supérieure, Paris
cousot@ens.fr  www.di.ens.fr/~cousot

Minta Martin Lecture, May 13th, 2005

Software is replacing humans
- Paris métro line 12 accident\(^1\): the driver was going **too fast**
- New high-speed métro line 14 (Météor): fully automated, no operators
- Software is in all **mission-critical and safety-critical** industrial infrastructures

\(^1\) On August 30th, 2000, at the Notre-Dame-de-Lorette métro station in Paris, a car flipped over on its side and slid to a stop just a few feet from a train stopped on the opposite platform (24 injured).
As computer hardware capacity grows... 

ENIAC
5,000 flops

NEC Earth Simulator
$35 \times 10^{12}$ flops

(1) Software gets huge

Software size grows...

Text editor
1,700,000 lines of C

Operating system
35,000,000 lines of C

Text editor
1,700,000 lines of C

Operating system
35,000,000 lines of C

1,700 bugs (estimation)

35,000,000 lines of C

3 months for full-time reading of the code
5 years for full-time reading of the code

... and so does the number of bugs

Text editor
1,700,000 lines of C

Operating system
35,000,000 lines of C

30,000 known bugs
(2) Computers are finite

Computers are finite
- Engineers use mathematics to deal with continuous, infinite structures (e.g. \( \mathbb{R} \))
- Computers can only handle discrete, finite structures

Putting big things into small containers
- Numbers are encoded onto a limited number of bits (binary digits)
- Some operations may overflow (e.g. integers: 32 bits \( \times 32 \text{ bits} = 64 \text{ bits} \))
- Using different number sizes (32, 64, \ldots \) bits) can also be the source of overflows

The Ariane 5.01 maiden flight
- June 4\textsuperscript{th}, 1996 was the maiden flight of Ariane 5
The Ariane 5.01 maiden flight failure

– June 4th, 1996 was the maiden flight of Ariane 5
– The launcher was destroyed after 40 seconds of flight because of a software overflow

6 A 16 bit piece of code of Ariane 4 had been reused within the new 32 bit code for Ariane 5. This caused an uncaught overflow, making the launcher uncontrollable.

(3) Computers go round

Modular arithmetic...

– Todays, computers avoid integer overflows thanks to modular arithmetic
– Example: integer 2’s complement encoding on 8 bits

... can be contrary to common sense

\[
\begin{align*}
# & 1073741823 + 1; \\
- & : \text{int} = -1073741824 \\
# & -1073741824 - 1; \\
- & : \text{int} = 1073741823 \\
# & -1073741824 ÷ -1; \\
- & : \text{int} =
\end{align*}
\]
... can be contrary to common sense

# 1073741823 + 1;
- : int = -1073741824
# -1073741824 - 1;
- : int = 1073741823
# -1073741824 \div -1;
- : int = -1073741824

Mapping many to few

- Reals are mapped to floats (floating-point arithmetic)
  \pm d_0.d_1d_2\ldots d_{p-1}\beta^e \)

- For example on 6 bits (with \( p = 3, \beta = 2, e_{\text{min}} = -1, e_{\text{max}} = 2 \)), there are 32 normalized floating-point numbers. The 16 positive numbers are

Rounding

- Computations returning reals that are not floats, must be rounded
- Most mathematical identities on \( \mathbb{R} \) are no longer valid with floats
- Rounding errors may either compensate or accumulate in long computations
- Computations converging in the reals may diverge with floats (and ultimately overflow)

Example of rounding error

/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
}
% gcc float-error.c
% /a.out
0.000000

/* double-error.c */
int main () {
  double x, y, z, r;
  x = 1.0exp(1.,50)+1.0exp(1.,26); /*
  y = 1125899973951488.0;
  z = x - 1;
  r = y - z;
  printf("%f\n", r);
}
% gcc double-error.c
% /a.out
134217728.000000

\((x + a) - (x - a) \neq 2a\)
Example of rounding error

/* float-error.c */
int main () {
    float x, y, z, r;
    x = 1.000000019e+38;
    y = x + 1.0e21;
    z = x - 1.0e21;
    r = y - z;
    printf("\%fn", r);
}

Example of accumulation of small rounding errors

% ocaml

Objective Caml version 3.08.1

# let x = ref 0.0;;
val x : float ref = {contents = 0.}

# for i = 1 to 1000000000 do
    x := !x +. 1.0/.10.0
done; x;;

- : float ref = {contents = 99999998.7454178184}

since (0.1)_{10} = (0.0001100110011001100...)_{2}

Explanation of the huge rounding error

1. **Floats**
   - **Reals**
     - \( x = 1.000000019 \times 10^{38} \)
     - Rounding
     - \( x \pm 10^{21} \)

2. **Doubles**
   - **Reals**
     - \( x = 1125899973951487.0 \)
     - **Floats**
     - \( x \pm 2^{-53} \)

(The Patriot missile failure)

- “On February 25th, 1991, a Patriot missile ... failed to track and intercept an incoming Scud.”

- The **software failure** was due to a cumulated rounding error.

---

8 This Scud subsequently hit an Army barracks, killing 28 Americans.
9 “Time is kept continuously by the system’s internal clock in tenths of seconds”
   “The system had been in operation for over 100 consecutive hours”
   “Because the system had been on so long, the resulting inaccuracy in the time calculation caused the range gate to shift so much that the system could not track the incoming Scud”
Warranty

Excerpt from an GPL open software licence:

NO WARRANTY. ... BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK OF A LOSS OF ANY KIND ARISING FROM THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

You get nothing for free!

What can be done about bugs?
Warranty

Excerpt from Microsoft software licence:

DISCLAIMER OF WARRANTIES. … MICROSOFT AND ITS SUPPLIERS PROVIDE THE SOFTWARE, AND SUPPORT SERVICES (IF ANY) AS IS AND WITH ALL FAULTS, AND MICROSOFT AND ITS SUPPLIERS HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY (IF ANY) IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FIT- NESS FOR A PARTICULAR PURPOSE, OF RELIABILITY OR AVAILABILITY, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF LACK OF NEGLIGENCE, ALL WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAIL- UR TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFT- WARE, AND RELATED CONTENT THROUGH THE SOFTWARE OR OTHER- WISE ARISING OUT OF THE USE OF THE SOFTWARE. …

You get nothing for your money either!

Mathematics and computers can help

– Software behavior can be mathematically formalized → semantics
– Computers can perform semantics-based program analyses to realize verification → static analysis
  - but computers are finite so there are intrinsic limi- tations → undecidability, complexity
  - which can only be handled by semantics approxima- tions → abstract interpretation

Abstract interpretation
(1) very informally

Traditional software validation methods

– The law cannot enforce more than “best practice”
– Manual software validation methods (code reviews, sim- ulations, tests, etc.) do not scale up
– The capacity of programmers/computer scientists re- mains essentially the same
– The size of software teams cannot grow significantly without severe efficiency losses
Operational semantics

Safety property

Test/debugging is unsafe

Abstract interpretation is safe
Soundness requirement: erroneous abstraction \(^\text{10}\)

**Forbiden zone**

**Erroneous trajectory abstraction**

**Possible trajectories**

---

Global interval abstraction → false alarms

**Forbidden zone**

**False alarms**

**Imprecise trajectory abstraction by intervals**

**Possible trajectories**

---

Imprecision → false alarms

**Forbidden zone**

**False alarm**

**Imprecise trajectory abstraction**

**Possible trajectories**

---

Local interval abstraction → false alarms

**Forbidden zone**

**False alarms**

**Imprecise trajectory abstraction by intervals**

**Possible trajectories**

---

\(^{10}\) This situation is always excluded in static analysis by abstract interpretation.
Refinement by partitionning

Intervals with partitionning

The ASTRÉE static analyzer

C programming language

with:
- boolean, integer & floating point computations
- pointers (on functions, etc), structures & arrays
- tests, loops and function calls
- limited branching (forward goto, break, continue)
without:
union, dynamic memory allocation, recursive function calls, unstructured backward branching, conflicting side effects\footnote{The ASTRÉE analyzer checks the absence of ambiguous side effects since otherwise the semantics of the C program would not be defined deterministically}
Operational semantics

- International norm of C (ISO/IEC 9899:1999)
- restricted by implementation-specific behaviors depending upon the machine and compiler
- restricted by user-defined programming guidelines
- restricted by program specific user requirements
- restricted by a volatile environment as specified by a trusted configuration file.

Implicit specification: absence of runtime errors

- No violation of the norm of C
- No implementation-specific undefined behaviors
- No violation of the programming guidelines
- No violation of the programmer assertions

Application domain

- Safety critical embedded real-time synchronous software for non-linear control of very complex control/command systems
- Strictly disciplined programming methodology
- 75% of the code is automatically generated from a high-level specification language
- The external controlled system is unknown (but for the range of a few volatile variables, maximal duration, ... as specified in the configuration file)

Verification of flight control software

- Primary flight control software of the Airbus A340 family and the A380 digital fly-by-wire systems
- Most critical software on board
- ASTRÉE verifies the absence of runtime errors without any false alarms!

controls automatically the airplane surface deflections and power settings, performs envelope protection, ... with precedence over the pilot.
Examples of abstractions in ASTRÉE

Ellipsoid Abstract Domain for Filters
- Computes $X_n = \left\{ \begin{array}{l} \alpha X_{n-1} + \beta X_{n-2} + Y_n \cr I_n \end{array} \right\}$
- The concrete computation is bounded, which must be proved in the abstract
- Polyhedral approximations are unstable
- The simplest stable surface is an ellipsoid

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)); }
    E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
    /* 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; }
}

Arithmetic-geometric progressions

// cat retro.c
typedef enum (FALSE=0, TRUE=1) BOOL;
BOOL FIRST;
volatile BOOL SWITCH;
volatile float E;
float P, X, A, B;
void dev () {
    X=E;
    if (FIRST) ( P = X; )
    else {
        P = (P - (((2.0 * P) - A) - B)
             + 4.91048e-03));
        B = A;
        if (SWITCH) (A = P;)
        else (A = X;)
    }
}
void main() {}
{ FIRST = TRUE;
    while (TRUE) {
        dev();
        FIRST = FALSE;
        _ASTREE_wait_for_clock();
    }
}
% cat retro.config
-ASTRÉE_volatile_input((E [-15.0, 15.0]));
-ASTRÉE_volatile_input((SWITCH [0,1]));
-ASTRÉE_max_clock((36000000));
P| <= (15. + 5.8774717541e-39)
/ 1.19209290217e-07 + (1 +
 1.19209290217e-07)\*clock -
5.8774717541e-39 / 1.19209290217e-07
class < 23.0393528881
Abstract interpretation (2) with a touch of formalism

Syntax of programs

\[
X \quad \text{variables } X \in X
\]
\[
T \quad \text{types } T \in T
\]
\[
E \quad \text{arithmetic expressions } E \in E
\]
\[
B \quad \text{boolean expressions } B \in B
\]
\[
D ::= T \; X; \\
| T \; X; \; D'
\]
\[
C ::= X = E; \\
| \text{while } B \; C' \\
| \text{if } B \; C' \text{ else } C'' \\
| \{ C_1 \ldots C_n \}, (n \geq 0)
\]
\[
P ::= D \; C
\]

Final states semantics

Semantics
States

Values of given type:

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

\[
\forall [\text{int}] = \{z \in \mathbb{Z} | \text{min\_int} \leq z \leq \text{max\_int}\}
\]

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

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

\[
\Sigma[T X;] = \{X\} \mapsto \forall[T]
\]

\[
\Sigma[T X; D] = (\{X\} \mapsto \forall[T]) \cup \Sigma[D]
\]

\[22 \text{ States } \rho \in \Sigma[P] \text{ of a program } P \text{ map program variables } X \text{ to their values } \rho(X)\]

Final states semantics

\[
S[X = B;]\ R \overset{\text{def}}{=} \{\rho[X \leftarrow \exists[B]\rho] \mid \rho \in R\}
\]

\[
\rho[X \leftarrow v](X) = v, \quad \rho[X \leftarrow v](Y) = \rho(Y)
\]

\[
S[\text{if } B C'' \text{ else } C']R = S[C'](B[B]R) \cup S[C''](B[-B]R)
\]

\[
B[B]R = \{\rho \in R \mid B \text{ holds in } \rho\}
\]

\[
S[\text{while } B C']R = \text{let } W = \text{lfp}\ C . R \cup S[C'](B[B]X)
\]

\[
in (B[-B]W)
\]

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

\[
S[C_1 \cdots C_n]R = S[C_1] \circ \cdots \circ S[C_n] \quad n > 0
\]

\[
S[D C]R = S[C](R) \quad (R \subseteq \Sigma[D], \text{ initial states})
\]

Undecidability

- The program's semantics, which is an infinite object, is not computable by a finite device

- All non-trivial questions about a program's semantics are undecidable (no computer can always answer, for sure, in a finite amount of time)

- Example: termination \[23\]

\[23 \text{ Assume } \text{termination}(P) \text{ is a terminating program answering correctly the following question about any program } P \text{ is a parameter encoded as text): Are all trajectories of } P \text{ finite?}\]

- A contradiction immediately appears when considering the program which tests:

  \[
  \text{program Goodel}(P); \text{ while termination}(P) \text{ do } \{} \text{ od}
  \]

- So termination is undecidable (whence so is any interesting semantic program property)
Polynomial Time Complexity

- **Polynomial-time computability** is identified with the intuitive notion of **algorithmic efficiency**.
- Intuitively valid only for small powers:

<table>
<thead>
<tr>
<th>$n$</th>
<th>$O(n)$</th>
<th>$O(n \cdot \log(n))$</th>
<th>$O(n^2)$</th>
<th>$O(n^3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>10</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>0.1 $\mu$s</td>
<td>1 $\mu$s</td>
</tr>
<tr>
<td>$10^3$</td>
<td>1 $\mu$s</td>
<td>6 $\mu$s</td>
<td>1 ms</td>
<td>1 s</td>
</tr>
<tr>
<td>$10^6$</td>
<td>1 ms</td>
<td>13 ms</td>
<td>16 mn</td>
<td>32 years</td>
</tr>
<tr>
<td>$10^9$</td>
<td>1 s</td>
<td>20 s</td>
<td>32 years</td>
<td>300 000 000 centuries</td>
</tr>
<tr>
<td>$10^{12}$</td>
<td>16 mn</td>
<td>7.7 h</td>
<td>300 000 centuries</td>
<td>—</td>
</tr>
<tr>
<td>$10^{15}$</td>
<td>11.6 days</td>
<td>1 year</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

- Execution time at $10^9$ ops/s

---

**Abstract interpretation**

- $\langle \varphi(\Sigma[P]), \subseteq \rangle \xrightarrow{\gamma} \langle L, \subseteq \rangle$
- $L$ encodes abstractions of properties in $\varphi(\Sigma[P])$
- $\subseteq$ abstracts implication $\subseteq$ \(^{24}\)
- $\alpha(I)$ encodes an overapproximation of property $I$ \(^{25}\)
- $\gamma(I)$ is the meaning of the abstract property $\overline{I}$
- Approximation is from above $I \subseteq \alpha \circ \gamma(I) \Rightarrow \overline{I}$
- In case of best approximation ($\alpha \circ \gamma(I) \subseteq \overline{I}$), $\langle \alpha, \gamma \rangle$ is a Galois connection

---

\(^{24}\) $\alpha$ and $\gamma$ order preserving
\(^{25}\) e.g. $\alpha$(set of points) = polyhedron and $\gamma$(polyhedron) = set of interior points
**Interval abstraction:**

**Polyhedral abstraction:**

---

**Examples**

**Abstract domain**

**Concrete domain**

**Abstract domain**

**Concrete domain**

---

**Function Abstraction**

\[ F^\# = \alpha \circ F \circ \gamma \]

\[ \langle P, \varnothing \rangle \xmapsto{\gamma} \langle Q, \varnothing \rangle \]

\[ \langle P \xmapsto{\text{mon}} P', \varnothing \rangle \xmapsto{\lambda F^\# \circ \alpha \circ F \circ \gamma} \langle Q \xmapsto{\text{mon}} Q', \varnothing \rangle \]

---

**Fixpoint abstraction**

**Approximation relation** \( \sqsubseteq \)

\[ F \circ \gamma \sqsubseteq \gamma \circ F^\# \Rightarrow \operatorname{lfp} F \sqsubseteq \gamma(\operatorname{lfp} F^\#) \]

---

**Abstract final state semantics**

\[ S^\#[X = E] \quad R \overset{\text{def}}{=} \alpha(\{ \rho \mid X \leftarrow E[R] \mid \rho \in \gamma(R) \}) \]

\[ S^\#[\text{if } B \ C' \text{ else } C''] \quad R \overset{\text{def}}{=} S^\#[C'][S^\#[B][R] \sqcup S^\#[C'']][B][\neg B][R] \]

\[ B^\#[B] \quad R \overset{\text{def}}{=} \alpha(\{ \rho \in \gamma(R) \mid B \text{ holds in } \rho \}) \]

\[ S^\#[\text{while } B \ C'] \quad R \overset{\text{def}}{=} \text{let } W = \operatorname{lfp} \lambda \chi \cdot R \sqcup S^\#[C'][B][\chi] \text{ in } (B^\#[\neg B][W]) \]

\[ S^\#[\varnothing] \overset{\text{def}}{=} R \]

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

\[ S^\#[D \ C] \overset{\text{def}}{=} S^\#[C](\alpha(R)) \quad \text{(initial states)} \]

The least fixpoint can be computed by elimination methods or by chaotic/asynchronous iteration methods but rapid convergence may not be guaranteed in infinite or very large abstract domains.
Abstract semantics with convergence acceleration

\[ S^\#[X = E; R] \overset{\text{def}}{=} \alpha(\{ \rho | X \leftarrow E; R \mid \rho \in \gamma(R) \}) \]

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

\[ S^\#[\text{while } B \text{ } C'; R] \overset{\text{def}}{=} \lambda X. \text{ let } \mathcal{F} = \lambda X. \text{ let } \mathcal{Y} = R \cup S^\#(B; C') \text{ in } \mathcal{Y} \subseteq X \text{ then } X \text{ else } X \rhd \mathcal{Y} \text{ and } \mathcal{W} = \text{up}^\mathcal{F} \text{ in } (B; \neg B) \mathcal{W} \]

\[ S^\#[\{\}] R \overset{\text{def}}{=} R \]

\[ S^\#[\{C_1 \ldots C_n\}; R] \overset{\text{def}}{=} S^\#(C_1) \circ \ldots \circ S^\#(C_n) \]

\[ S^\#[D; C; R] \overset{\text{def}}{=} S^\#(C)(\alpha(R)) \] (initial states)

---

Applications of Abstract Interpretation

Abstract interpretation formalizes sound approximations as found everywhere in computer science:

- Syntax Analysis [TCS 290(1) 2002]
- Hierarchies of Semantics (including Proofs) [POPL ’92], [TCS 277(1–2) 2002]
- Program Transformation [POPL ’02]
- Typing & Type Inference [POPL ’97]
- (Abstract) Model Checking [POPL ’00]
Applications of Abstract Interpretation (Cont’d)

- **Bisimulations** [RT-ESOP ’04]
- **Software Watermarking** [POPL ’04]
- **Code obfuscation** [DPG-ICALP ’05]
- **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], …
  - **WCET** [EMSOFT ’01], …

**Project while visiting MIT**

**Computer controlled systems**

**Software analysis & verification**

Abstractions: program → precise, system → coarse
System analysis & verification

Abstractions: program → precise, system → precise

Grand challenge

Software verification
– is the grand challenge for computer scientists and engineers in the next 15 years
– will not be convincing without global system verification

Conclusion

THE END

My MIT web site is www.mit.edu/~cousot, where these slides are available
My ENS web site is www.di.ens.fr/~cousot
For more technical details, see the MIT course 16.399 on Abstract interpretation
web.mit.edu/16.399/
References

[1] www.astree.ens.fr [3, 4, 5, 7, 8, 9, 10]


