Automatic Verification of Avionic Synchronous Safety Critical Embedded Software

Patrick COUSOT
École Normale Supérieure
45 rue d’Ulm
75230 Paris cedex 05, France
Patrick.Cousot@ens.fr
www.di.ens.fr/~cousot

Wednesday Seminar Series, The Computer Laboratory
University of Cambridge, UK, Oct. 20th, 2004

Motivation

— 3 —

All Computer Scientists Have Experienced Bugs

It is preferable to verify that safety-critical programs do not go wrong before running them.
Static Analysis by Abstract Interpretation

**Static analysis**: analyse the program at compile-time to verify a program runtime property (e.g. the absence of some categories of bugs)

**Undecidability**

**Abstract interpretation**: effectively compute an abstraction/sound approximation of the program semantics,
- which is **precise** enough to imply the desired property, and
- coarse enough to be **efficiently computable**.

---

**Abstract Interpretation, Informally**

---

**Operational Semantics**

**Safety property**

---

**Forbidden zone**
Test/Debugging is Unsafe

Forbidden zone

Error !!!

Possible trajectories

Test of a few trajectories

x(t)

Bounded Model Checking is Unsafe

Forbidden zone

Error !!!

Possible trajectories

Bounded model-checking of trajectory prefixes

x(t)

Abstract Interpretation

Forbidden zone

Possible trajectories

Abstraction of the trajectories

x(t)

Soundness: Erroneous Abstraction

Forbidden zone

Error !!!

Possible trajectories

Erroneous trajectory abstraction

x(t)
Soundness: Erroneous Abstraction — II

Interval Abstraction ⇒ False Alarms

Imprecision ⇒ False Alarms

Refinement by Partitionning
Concrete Semantic Domain of Programs

Reachability properties:

\[ \Sigma[D C] \stackrel{\text{def}}{=} \Sigma[D] \]
\[ \Sigma[T X ; ] \stackrel{\text{def}}{=} \{ X \} \mapsto T \]
\[ \Sigma[T X ; D] \stackrel{\text{def}}{=} (\{ X \} \mapsto T) \cup \Sigma[D] \]
\[ D[P] \stackrel{\text{def}}{=} \rho(\Sigma[P]) \]

states \( \rho \)

(\( \rho(X) \) is the value of \( X \))

sets of states/

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

Concrete Reachability Semantics of Programs

\[ S[X = E ; ]R \stackrel{\text{def}}{=} \{ \rho[X \leftarrow E] : \rho \in R \cap \text{dom}(E) \} \]
\[ \rho[X \leftarrow v] (X) = v, \quad \rho[X \leftarrow v](Y) = \rho(Y) \]
\[ S[\text{if } B \ C']R \stackrel{\text{def}}{=} S[C'](B[R] \cup B[\neg B]R) \]
\[ B[R] \stackrel{\text{def}}{=} \{ \rho \in R \cap \text{dom}(B) : B \text{ holds in } \rho \} \]
\[ S[\text{if } B \ C' \text{ else } C'']R \stackrel{\text{def}}{=} S[C'](B[R] \cup S[C''](B[R] \cup B[\neg B]R)) \]
\[ S[\text{while } B \ C']R \stackrel{\text{def}}{=} \text{let } \lambda X. R \cup S[C'](B[R] X) \text{ in } (B[\neg B]W) \]
\[ S[\{ \} ]R \stackrel{\text{def}}{=} R \]
\[ S[\{C_1 \ldots C_n\}]R \stackrel{\text{def}}{=} S[C_n] \circ \ldots \circ S[C_1] \quad n > 0 \]
\[ S[D C]R \stackrel{\text{def}}{=} S[C]((\Sigma[D]) \quad \text{(uninitialized variables)} \]

Not computable (undecidability)
Example: Abstract Semantic Domain of Programs

\[ \langle \mathcal{D}^\#[P], \subseteq, \bot, \sqcup \rangle \]

such that:

\[ \langle \mathcal{D}, \subseteq \rangle \xrightarrow{\gamma} \langle \mathcal{D}^\#[P], \subseteq \rangle \]

hence \( \langle \mathcal{D}^\#[P], \subseteq, \bot, \sqcup \rangle \) is a complete lattice such that \( \bot = \alpha(\emptyset) \) and \( \sqcup X = \alpha(\cup \gamma(X)) \)

---

Convergence Acceleration with Widening

Abstract Reachability Semantics of Programs

\[
\begin{align*}
S^\#[X = E;]R & \overset{\text{def}}{=} \alpha(\{\rho[X \leftarrow E[R] \mid \rho \in \gamma(R) \cap \text{dom}(E)\}) \\
S^\#[\text{if } B C' \text{ else } C'']R & \overset{\text{def}}{=} S^\#[C'][B][R] \cup S^\#[\neg B][R] \\
B^\#[B][R] & \overset{\text{def}}{=} \alpha(\{\rho \in \gamma(R) \cap \text{dom}(B) \mid B \text{ holds in } \rho\}) \\
S^\#[\text{while } B C' \text{ } R] & \overset{\text{def}}{=} \text{let } \mathcal{W} = \text{lfp}_\gamma \lambda X. R \cup S^\#[C'][B][R] \text{ in } (B^\#[\neg B][\mathcal{W}]) \\
S^\#[\{\}]R & \overset{\text{def}}{=} R \\
S^\#[C_1 \ldots C_n]R & \overset{\text{def}}{=} S^\#[C_n] \circ \ldots \circ S^\#[C_1] \quad n > 0 \\
S^\#[D C]R & \overset{\text{def}}{=} S^\#[C](T) \quad (\text{uninitialized variables}) \\
\end{align*}
\]
Example: Abstract Semantics with Convergence Acceleration

\[ S^\#[X = E; R] \equiv \alpha(\{\rho[X \leftarrow E[\rho] | \rho \in \gamma(R) \cap \text{dom}(E)\}) \]
\[ S^\#[\text{if } B \text{ } C'] R \equiv S^\#[C'](B^\#[B]R) \cup B^\#[\neg B]R \]
\[ B^\#[B] R \equiv \alpha(\{\rho \in \gamma(R) \cap \text{dom}(B) | B \text{ holds in } \rho\}) \]
\[ S^\#[\text{if } B \text{ } C' \text{ else } C'' R] \equiv S^\#[C'](B^\#[B]R) \cup S^\#[C''](B^\#[\neg B]R) \]
\[ S^\#[\text{while } B \text{ } C'] R \equiv \text{let } F^\# = \lambda X. \text{let } Y = R \cup S^\#[C'](B^\#[B]X) \text{ in if } Y \subseteq X \text{ then } X \text{ else } X \bigtriangledown Y \text{ and } W = \text{lfp}_\bot \text{ in } (B^\#\neg B^\#W) \]
\[ S^\#[\text{while } B \text{ } C'] R \equiv \text{lfp}_\bot \text{ in } (B^\#\neg B^\#W) \]

Applications of Abstract Interpretation

- **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]

Applications of Abstract Interpretation (Cont’d)

- (Abstract) Model Checking [POPL ’00]
- **Program Transformation** [POPL ’02]
- **Software Watermarking** [POPL ’04]
- **Bisimulations** [RT-ESOP ’04]

All these techniques involve sound approximations that can be formalized by abstract interpretation
A Practical Application of Abstract Interpretation to the Verification of Safety Critical Embedded Control-Command Software

Reference


ASTRÉE: A Sound, Automatic, Specializable, Domain-Aware, Parametric, Modular, Efficient and Precise Static Program Analyzer

www.astree.ens.fr

- C programs:
  - with
    * pointers (including on functions), structures and arrays
    * floating point computations
    * tests, loops and function calls
    * limited branching (forward goto, break, continue)

- without
  - union
  - dynamic memory allocation
  - recursive function calls
  - backward branching
  - conflict side effects
  - C libraries

- Application Domain: safety critical embedded real-time synchronous software for non-linear control of very complex control/command systems.

Concrete Operational Semantics

- International norm of C (ISO/IEC 9899:1999)
- restricted by implementation-specific behaviors depending upon the machine and compiler (e.g. representation and size of integers, IEEE 754-1985 norm for floats and doubles)
- restricted by user-defined programming guidelines (such as no modular arithmetic for signed integers, even though this might be the hardware choice)
- restricted by program specific user requirements (e.g. assert)
Abstract Semantics

- **Reachable states** for the concrete operational semantics
- **Volatile environment** is specified by a trusted configuration file.

---

Implicit Specification: Absence of Runtime Errors

- No violation of the **norm of C** (e.g. array index out of bounds)
- **No implementation-specific undefined behaviors** (e.g. maximum short integer is 32767)
- No violation of the **programming guidelines** (e.g. static variables cannot be assumed to be initialized to 0)
- No violation of the **programmer assertions** (must all be statically verified).

---

Example application

- **Primary flight control software** of the Airbus A340/A380 fly-by-wire system
- C program, automatically generated from a proprietary high-level specification (à la Simulink/SCADE)
- A340 family: 132,000 lines, 75,000 LOCs after preprocessing, 10,000 global variables, over 21,000 after expansion of small arrays
- A380: × 3

---

The Class of Considered Periodic Synchronous Programs

```c
declare volatile input, state and output variables;
initialize state and output variables;
loop forever
  - read volatile input variables,
  - compute output and state variables,
  - write to volatile output variables;
  wait_for_clock();
end loop
```

- **Requirements:** the only interrupts are clock ticks;
- **Execution time of loop body less than a clock tick** [3].

---

Reference

Characteristics of the ASTRÉE Analyzer

**Static:** compile time analysis (≠ run time analysis Rational Purify, Parasoft Insure++)

**Program Analyzer:** analyzes programs not micromodels of programs (≠ PROMELA in SPIN or Alloy in the Alloy Analyzer)

**Automatic:** no end-user intervention needed (≠ ESC Java, ESC Java 2)

**Sound:** covers the whole state space (≠ MAGIC, CBMC) so never omit potential errors (≠ UNO, CMC from coverity.com) or sort most probable ones (≠ Splint)

---

Characteristics of the ASTRÉE Analyzer (Cont’d)

**Multiabstraction:** uses many numerical/symbolic abstract domains (≠ symbolic constraints in Bane or the canonical abstraction of TVLA)

**Infinitary:** all abstractions use infinite abstract domains with widening/narrowing (≠ model checking based analyzers such as VeriSoft, Bandera, Java PathFinder)

**Efficient:** always terminate (≠ counterexample-driven automatic abstraction refinement BLAST, SLAM)

**Specializable:** can easily incorporate new abstractions (and reduction with already existing abstract domains) (≠ general-purpose analyzers PolySpace Verifier)

**Domain-Aware:** knows about control/command (e.g. digital filters) (as opposed to specialization to a mere programming style in C Global Surveyor)

**Parametric:** the precision/cost can be tailored to user needs by options and directives in the code

---

Characteristics of the ASTRÉE Analyzer (Cont’d)

**Automatic Parametrization:** the generation of parametric directives in the code can be programmed (to be specialized for a specific application domain)

**Modular:** an analyzer instance is built by selection of OCAML modules from a collection each implementing an abstract domain

**Precise:** very few or no false alarm when adapted to an application domain → it is a VERIFIER!
Example of Analysis Session

(Benchmarks (Airbus A340 Primary Flight Control Software))

- 350,000 lines
- 0 alarms (last week!),
  7h on 2.8 GHz PC,
  1 Gigabyte
  \rightarrow A world grand première!

---

Examples of Abstractions

- 132,000 lines, 75,000 LOCs after preprocessing
- Comparative results (commercial software):
  4,200 (false?) alarms,
  3.5 days;
- Our results:
  0 alarms,
  40mn on 2.8 GHz PC,
  300 Megabytes
  \rightarrow A world première!

---

We are still in a phase where we favour precision rather than computation costs, and this should go down.
For example, the A340 analysis went up to 5 h, before being reduced by requiring less precision while still getting no false alarm.
General-Purpose Abstract Domains: Intervals and Octagons

Intervals:
\[
\begin{align*}
1 \leq x & \leq 9 \\
1 \leq y & \leq 20
\end{align*}
\]

Octagons [4]:
\[
\begin{align*}
1 \leq x & \leq 9 \\
x + y & \leq 77 \\
1 \leq y & \leq 20 \\
x - y & \leq 04
\end{align*}
\]

Difficulties: many global variables, arrays (smashed or not), IEEE 754 floating-point arithmetic (in program and analyzer) [5]

Symbolic abstract domain

- Interval analysis: if \( x \in [a, b] \) and \( y \in [c, d] \) then \( x - y \in [a - c, b - d] \) so if \( x \in [0, 100] \) then \( x - x \in [-100, 100] \)!!
- The symbolic abstract domain propagates the symbolic values of variables and performs simplifications;
- Must maintain the maximal possible rounding error for float computations (overestimated with intervals);

Floating-Point Computations

\((x + a) - (x - a) \neq 2a\)

Code Sample:

```c
/* 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
", r);
} % gcc float-error.c
% ./a.out
```
**Boolean Relations for Boolean Control**

**Code Sample:**

```c
/* boolean.c */
typedef enum {F=0, T=1} BOOL;
BOOL B;
void main () {
    unsigned int X, Y;
    while (1) {
        ...
        B = (X == 0);
        ...
        if (!B) {
            Y = 1 / X;
        }
        ...
    }
}
```

The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leaves.

**Control Partitionning for Case Analysis**

**Code Sample:**

```c
/* trace_partitionning.c */
void main() {
    float t[5] = {-10.0, -10.0, 0.0, 10.0, 10.0};
    float c[4] = {0.0, 2.0, 2.0, 0.0};
    float d[4] = {-20.0, -20.0, 0.0, 20.0};
    float x, r;
    int i = 0;
    ...
    while ((i < 3) && (x >= t[i+1])) {
        i = i + 1;
    }
    r = (x - t[i]) * c[i] + d[i];
}
```

Delivering abstract unions in tests and loops is more precise for non-distributive abstract domains (and much less expensive than disjunctive completion).

**2nd Order Digital Filter:**

```c
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;
    }
}
```

**Ellipsoid Abstract Domain for Filters**

- Computes $X_n = \frac{\alpha X_{n-1} + \beta X_{n-2} + Y_n}{I_n}$
- 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.

**Filter Example [6]**

(Automatic) Parameterization

- All abstract domains of ASTRÊE are parameterized, e.g.
  - variable packing for octagones and decision trees,
  - partition/merge program points,
  - loop unrollings,
  - thresholds in widenings, . . . ;
- End-users can either parameterize by hand (analyzer options, directives in the code), or
- choose the automatic parameterization (default options, directives for pattern-matched predefined program schemata).

Possible origins of imprecision and how to fix it

In case of false alarm, the imprecision can come from:
- Abstract transformers (not best possible) —→ improve algorithm;
- Automatized parametrization (e.g. variable packing) —→ improve pattern-matched program schemata;
- Iteration strategy for fixpoints —→ fix widening \(^3\);
- Inexpressivity i.e. indispensable local inductive invariant are inexpressible in the abstract —→ add a new abstract domain to the reduced product (e.g. filters).

---

The main loop invariant for the A340

A textual file over 4.5 Mb with
- 6,900 boolean interval assertions \((x \in [0; 1])\)
- 9,600 interval assertions \((x \in [a; b])\)
- 25,400 clock assertions \((x + \text{clk} \in [a; b] \land x - \text{clk} \in [a; b])\)
- 19,100 additive octagonal assertions \((a \leq x + y \leq b)\)
- 19,200 subtractive octagonal assertions \((a \leq x - y \leq b)\)
- 100 decision trees
- 60 ellipse invariants, etc . . .

involving over 16,000 floating point constants (only 550 appearing in the program text) \(\times 75,000\) LOCs.

---

\(^3\) This can be very hard since at the limit only a precise infinite iteration might be able to compute the proper abstract invariant. In that case, it might be better to design a more refined abstract domain.
Conclusion

- Most applications of abstract interpretation tolerate a small rate (typically 5 to 15%) of false alarms:
  - Program transformation → do not optimize,
  - Typing → reject some correct programs, etc,
  - WCET analysis → overestimate;
- Some applications require no false alarm at all:
  - Program verification.
- Theoretically possible [SARA '00], practically feasible [PLDI '03]

References


The Future & Grand Challenges

Future (5 years):
- Asynchronous concurrency (for less critical software)
- Functional properties (reactivity)
- Industrialization

Grand challenge:
- Verification from specifications to machine code (verifying compiler)
- Verification of systems (quasi-synchrony, distribution)


