Static analysis: from theory to practice

David Monniaux

CNRS / VERIMAG

A joint laboratory of CNRS,
Université Joseph Fourier (Grenoble) and Grenoble-INP.

19 juin 2009

HDR presented before

"Rapporteurs"

Pr Roberto Giacobazzi (U. Verona)
Pr Eric Goubault (École polytechnique / CEA)
Pr Andreas Podelski (U. Freiburg)

Other members

Pr Gilles Dowek (École polytechnique)
Pr Roland Groz (Grenoble-INP)
Pr Helmut Seidl (TU-München)

Research area in a nutshell

Program analysis

Program (source code or object code)
$\Downarrow$ automatic analysis
Facts about the program

Connex areas

Semantics of real-life programming languages (floating-point, asynchronous parallelism)
Use of proof assistants
Decision of logic theories, quantifier elimination

Flashback: PhD thesis

2001 PhD thesis on the notion of abstract interpretation of probabilistic programs (semantics as Markov chains, Markov decision processes).

Mostly theoretical.

Plan

The Team

"Semantics and abstract interpretation" team at LIENS

Bruno Blanchet
Patrick Cousot (project leader)
Radhia Cousot
Jérôme Feret
Laurent Mauborgne
Antoine Miné
yours truly
Xavier Rival

Plan

1996: Ariane 501

Before

After

This event raised great awareness of the consequences of bugs in critical systems. Earlier accidents (e.g. Therac-25 radiotherapy machine, killed patients) had less publicity.

Fly-by-wire controls: Airbus A380

Exemplified by sidesticks: commands from pilots get sent to computers, which control the active surfaces.

Fly-by-wire controls: Boeing 777-200ER

The control yoke is “fake”: it is not mechanically tied to the active surfaces, but to a computer.

Airbus: a long-standing investment in quality software

Airbus develops control-command software using high-level tools such as SAO and SCADE (Lustre).

Airbus uses verification techniques (e.g. the CAVEAT Hoare logic proof assistant).

Airbus uses abstract interpretation

- AbsInt's worst-case execution time analysis tool.
- CEA's Fluctuat etc. tools.
- Astrée

Plan

The challenge

Prove the absence of runtime errors in critical code (from C source):

- overflow (see e.g. Ariane 501)
- divide by zero
- array access out of bound
- bad pointer
- other undefined behaviours

Challenges:

- very large code
- high precision requested (few false alarms)
- lots of floating-point

Difficulties

High precision

In static analysis for optimization, 95% completeness (95% of code without any alarms shown) is marvellous.

In static analysis for certification, this is very bad. On a 100,000 line program, this means 5,000 warnings — far too many for engineers!

C language and floating-point

Somewhat fuzzy and complex semantics. Thus not so-well studied by scientists, who prefer "nicer" cases.

Static analysis by abstract interpretation

Reachability analysis

$X_0 =$ initial states

$X_n =$ states reachable in $\leq n$ steps

$X_{n+1} = \phi(X_n)$ where $X = X_0 \cup f(X_n)$

$X_\infty =$ reachable states $= \lim_{n\to\infty} X_n = \bigcup_n X_n$.

Abstract interpretation

Replace X_n (difficult to represent exactly) by overapproximation $\gamma(X_n^\sharp)$ where $X_n^\sharp$ machine representable.

Example : $X_n^\sharp$ pair (a, b) defining the interval $\gamma(X_n^\sharp) = [a, b]$.

If needed, accelerate convergence using widening operators.

Concrete results

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

Absence of runtime errors, no/few false alarms in:

- 2003 Primary fly-by-wire control A340
- 2005 Fly-by-wire control A380
- 2008 Automatic docking software of the ATV (automatic transfer vehicle for International space station)

(and other industrial programs)

How these results were achieved

- Development of specific abstract domains.
- Maximizing synergies between abstract domains.
- Coarse analysis as long as its coarseness is not a problem.
- Careful use of data structures (large programs = large memory = any weakness may cause inacceptably large analysis time or memory usage).

..and research and innovations on little-studied areas of static analysis.

Semantics of C and floating-point

Difficulties:

- Memory model. Common assumptions outside of C specification.
- Interactions between C and IEEE-754 floating-point (a mess — see TOPLAS '08 article).

These issues need careful practical studies. Standards are fuzzy and anyway compilers do not respect them.

Some assumptions made by some proof tools are actually unsound.

Floating-point and numerics

Misleading prejudice

- "Floating-point is just like reals." (optimistic)
- "Floating-point can yield just about anything, it's so unpredictable." (pessimistic)
- "Floating-point programs run identically on all IEEE-754 systems." (optimistic)
- "Floating-point is too complicated for analysis." (pessimistic)

My opinion

- Floating-point is partially specified, and what is specified is enough to derive bounds that can prove many properties in programs written by normal people.
- Programs relying on finer properties not provable using these bounds should only be written by experts (e.g. William Kahan).

Floating-point and numerics analysis

Floating-point

Floating-point = exact real computation + boundable error [Miné]:
floating-point semantics $\sqsubseteq$ real nondeterministic semantics
$\sqsubseteq$ real (ideal) analysis $\sqsubseteq$ float implementation of analysis

Filtering

Numerical filtering: in control-command theory books, bounds on floating-point roundoff errors ignored ("useless" or "too pessimistic").

Results by Feret and Monniaux: any linear filter implemented in floating-point can be abstracted as $O = R.I + \epsilon$, $|\epsilon| \leq E.|I|$ (R ideal impulse response expressed as its Z-transform using rational functions). [CAV '05]

Low-level device drivers

Modern hardware controllers are programmable and transfer data using bus-master DMA.

- Positive point: offload work from the CPU to the controller, acts quite autonomously, does not need interrupts and reprogramming all the time.
- Negative point: more like a shared-memory multi-CPU system than a single-CPU system.

Contribution: attempt at modelization of USB OHCI controller as an asynchronous process composed with the driver ran on CPU [EMSOFT '07].

All other driver analyses ignore devices, some even ignore other threads.

Parallelization of analysis

Exploit the structure of the program to be analyzed in order to derive independently analyzable parts.

Synchronous code with multiple clocks statically scheduled:

```
phase = 0;
while (true) {
  executePhase(phase);
  phase = phase+1;
  if (phase==10) phase=0;
}
```

Parallelization of analysis (2)

Abstract as:

```
while (true) {
  phase = [0, 9];
  executePhase(phase);
}
```

Run the analysis of all phases in parallel (or grouped in chunks, per number of CPUs).

What I got out of it

For analysis designers

- It is possible to design static analyzers with very low levels of false alarms by targeting specific domains (e.g. control-command codes written in a certain way).
- This requires significant work. Orders of magnitude more than for toy languages and examples.
- We academics should take into account the difference between "clean" models and the actual systems they are meant to represent.

For industrials

- Static analysis is not a replacement for testing.
- Static analysis should be integrated into the development process early on, not as an afterthought.

Plan

A case for formally proved tools

We "prove" properties of programs...

- But how about bugs in the compiler? (see work by e.g. X. Leroy, X. Rival)
- How about bugs in the analyzer?

Formalizations of elements of static analyzers in Coq:

- The balanced tree data structure used in Astrée, with correctness proofs.
- "Minimalistic" notion of widening [under subm. HOSC].

Plan

Known weaknesses of many abstract interpretation schemes

- Lack of modularity (but see e.g. Logozzo's class-invariant analysis)
  - must re-run analysis if some part changes
  - cannot analyze program fragments with environment abstracted away
- Widening operators
  - no guarantee of precision
  - non-monotonic results
  - design somewhat of a black art

What can we do?

Direct computation of least invariants

I is an inductive invariant iff it contains the initial state and it is stable by the denotational semantics of the program:

$$x_0 \in I \tag{1}$$

$$\forall x \forall x' \; I(x) \wedge [\![P]\!](x, x') \Rightarrow I(x') \tag{2}$$

Suppose

- the state x is a tuple $\langle x_1, \ldots, x_n \rangle$ of reals
- I is a conjunction of $P_i(x_1, \ldots, x_n) \leq C_i$, $1 \leq i \leq m$ where the P_i are fixed polynomials and the C_i are free variables (parameters).

Then Eqn. 1 is a formula with free variables $C_1, \ldots, C_m$ whose solutions are the shape parameters defining inductive invariants.

Set of solutions

Example

Choose x_p in $[a, b]$
Infinite loop:
  Choose x in $[a, b]$
  If $x > x_p + \delta$ then $x := x_p + \delta$
  If $x < x_p - \delta$ then $x := x_p - \delta$
  $x_p := x$

Find invariants of the form $x_p \in [c, d]$. The condition becomes:

$$\forall x_p \; a \leq x_p \leq b \Rightarrow c \leq x_p \leq d \wedge$$
$$\forall x_p \; c \leq x_p \leq d \; \forall x, x', x'' \; (x > x_p + \delta \; ? \; x' = x_p + \delta : x' = x)$$
$$\wedge \, (x' < x_p - \delta \; ? \; x'' = x_p - \delta : x'' = x') \Rightarrow c \leq x'' \leq d \quad (3)$$

where $p \; ? \; xa : b$ is short for $(p \wedge a) \vee (\neg p \wedge b)$.

Quantifier elimination yields least invariants

The above formula can be simplified as $c \leq a \wedge d \geq b$.

If we ask for minimal d and maximal c (least invariant) the formula simplifies to $c = a \wedge d = b$.

But how can we do this automatically? By quantifier elimination [SAS '07, POPL '09].

From a formula with quantifiers, output an equivalent formula without quantifiers.

Work on quantifier elimination

Past state of the art

- The nonlinear problem over the reals is very hard (CAD, best known algorithm, is very complex to implement and slow).
- Known algorithms for the linear problem (LRA) were also slow.

Work

- Developed a quantifier elimination for LRA based on SMT-solving (SAT modulo the LRA theory) and eager constraint generation [LPAR '08].
- Implemented it in a tool.
- Developed a lazy version and implemented it, currently benchmarking it.

SMT-solving: use of numerical techniques

Most proof assistants / analysis tools / decision procedures use exact arithmetic, which can be costly. Can we use numerical methods?

Polynomial inequalities

Finding Positivstellensatz witnesses reduced to semidefinite programming (SDP).
Solve SDP using numerical methods.
Does not work well due to fatal geometric degeneracies (hard to get rid of).

Linear inequalities

- Floating-point simplex for exact computations: interesting efficiency in some cases [CAV '09].
- Interior point method for the witness problem: remains to be tested (degeneracy problem easier than for Positivstellensatz).

Plan

Short and medium term

Current

- Work with graduate student Julien Le Guen, Nicolas Halbwachs and STMicroelectronics on using static analysis for optimization.
- Work on quantifier elimination (lazy strategy).
- Work on numerical methods for finding invariants or checking formulas.

Medium term

- Possibly work on extracting formal proofs.
- Application to the modular analysis of LUSTRE?
- Another round at analysis of C?

Industrial lessons learned

- There are many interesting and hard research problems arising from industrial needs.
- Some industrial difficulties are actually self-inflicted organisational issues, and are *not* in need of a technical solution.
- There is a vast difference between what researchers like to research (clean semantics, mathematical structures, etc.) and real-life languages and systems.
- It is more reasonable to start with modest goals (domain-specific static analysis) than to target all kinds of programs.
- Verification (= proof of absence of bugs) is not at all the same as bug-finding.

Research lessons learned

- There are many interesting techniques coming from operational research, signal processing, automatic control theory...
- Yet the need for sound proofs often makes them unsuitable for direct application in formal methods.
- Sometimes, these techniques can be used with some adaptation (bounding ε's, checking phase in exact precision, etc.).

Questions