The Astrée static analyzer and beyond

David Monniaux

http://www.di.ens.fr/~monniaux

Centre National de la Recherche Scientifique

École Normale Supérieure

Département d'Informatique

45, rue d'Ulm, 75230 Paris cedex 5, France

Introduction

The Astrée static analyzer

Astrée `http://www.astree.ens.fr` is a static analyzer based on abstract interpretation.

- Analyzes a subset of the C language.
- Machine integers and floating-point numbers, not "mathematical" integers and real numbers.
- Tuned for large-scale control/command codes, automatically generated from high-level specifications.
- Precise domains for numerical computations.
- Detects runtime errors.

Challenges

Has to analyze the original source code, not a derived “model”.

Has to be sound (i.e. not say “there is no runtime error possible” when there are)

Has to be precise (i.e. not warn about many *possible* alarms that can’t happen — false alarms)

Handle floating-point well, including digital filtering algorithms

The biggest challenge

Very large software ($\gg$ 300,000 LOC) $\Rightarrow$ efficiency questions !

Commodity PC hardware $\Rightarrow$ keep memory requirements low

$\Rightarrow$ keep analysis times low

Efficiency considerations

False "good idea" For final "certification" of the system, only need a single pass of analysis, even if it is slow.

In reality... You want fast analysis

- for debugging the analyzer
- for using it while you develop the analyzed code
- for debugging input specifications (i.e. bounds on the inputs)

The team

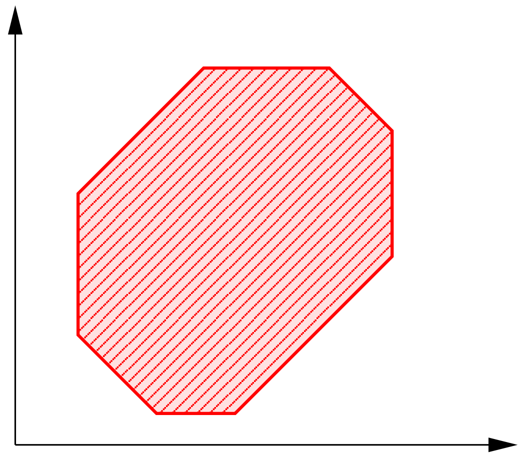
Bruno Blanchet [CNRS] (formerly)
Patrick [ENS] and Radhia Cousot [CNRS]
Jerôme Feret [ENS]
Laurent Mauborgne [ENS]
Antoine Miné [ENS]
David Monniaux [CNRS]
Xavier Rival [ENS] (now postdoc at Berkeley)

Abstract interpretation

Basic idea

To each program point, set of reachable memory (variable) states at this point

Over-approximate the set by some constraints



Abstract operations

A program statement has semantics $[\![P]\!] : \mathcal{P}(X) \rightarrow^{\cup} \mathcal{P}(X)$ ($X =$ possible memory states).

$\gamma(x)$: x abstraction (ex: shape of polyhedron), $\gamma(x)$ set of concrete memory states represented

Abstraction of a program statement: $[\![P]\!]^{\sharp} : X^{\sharp} \rightarrow X^{\sharp}$

Soundness: $[\![P]\!] \circ \gamma(x^{\sharp}) \subseteq \gamma \circ [\![P]\!]^{\sharp} (x^{\sharp})$

Tests

$[\![B]\!]$ = set of states matched by boolean expression B

Abstract $W \mapsto W \cap [\![B]\!]$

Can be constructed from tests for atomic expressions and $\sqcup$

$\gamma(a^\sharp) \cup \gamma(b^\sharp) \sqsubseteq \gamma(a \sqcup b)$

Ex: polyhedra $\Rightarrow$ convex hull

$$[\![\texttt{if } c \texttt{ then } p_1 \texttt{ else } p_2]\!](x) = [\![p_1]\!](x \cap [\![c]\!]) \cup [\![p_2]\!](x \cap [\![c]\!]^C)$$
$$[\![\texttt{if } c \texttt{ then } p_1 \texttt{ else } p_2]\!]^\sharp(x^\sharp) = [\![p_1]\!]^\sharp \circ [\![c]\!]^\sharp(x^\sharp) \sqcup [\![p_2]\!]^\sharp \circ [\![\neg c]\!]^\sharp(x^\sharp)$$

System of equations approach

Express the set of states at each statement as a function of the set of states at other statements.

```
  beep
A:
  foo
  bar
  if (xyz) goto B;
C:goto A;
```

States at point `A`: outcome from `beep` $\cup$ states at point `C`

What is the concrete fixed point?

Sets of reachable states at program points:

$$S_1 = F_1(S_1, S_2, \ldots, S_n) \quad (1)$$

$$\vdots \quad (2)$$

$$S_n = F_n(S_1, S_2, \ldots, S_n) \quad (3)$$

$$\quad (4)$$

All F_i functions ω-continuous

Start at $\perp$ and iterate ω times to reach fixed point.

Abstract fixed point

Replace all F_i by $F_i^\sharp$.

Accelerate convergence using a widening operator ∇

Concrete sequence $x_1, x_2, x_3 \dots$ approximated by $x_1^\sharp, x_2^\sharp, x_3^\sharp \dots$ with $\forall i \; x_i \subseteq \gamma(x_i^\sharp)$.

In practice

Iterations use the block-structure of the source code

Saves memory: one abstract environment per nested loop

Trace partitioning

An open system

Simple I/O

System takes scalar values (integers, floats) as inputs
Simple interval constraints known on them: input cannot go beyond [-10, 10]

Prove "worst case" behaviour

Physical feedback

Industrial users would like user-level properties such as "in normal conditions, FOOBAR_X is in [-30, 30]"

At present Astrée does worst case behavior, say [-1000, 1000]

Reconstructed scenarios of inputs are often physically impossible

Future work: How about an abstract model of the physical world? (Dynamic system, real numbers)

Intelligent I/O

Previous analyses: system does I/O through simple memory-mapped registers, aka `volatile` variables

Increasing use of intelligent I/O controllers, coprocessors executing I/O programs (industrial req.)

Example: USB controller, OHCI spec, asynchronously updates linked lists (for worklists) and does bus-mastering DMA

Only vague specification for I/O controller (OHCI spec = book in English, many details somewhat vague) ⇒ model as highly nondeterministic routines

Asynchronous model

One main big synchronous process
Some known memory-mapped I/O zones (if necessary through preanalysis)
Some small nondeterministic routines processed asynchronously (e.g. "perhaps process one item from the worklist")

Semantics: before each time when the main process reads/writes MMIO or DMA zones, asynchronous routines may execute an unknown number of times

Each async routine implements some atomic step of the I/O controller (e.g. "transmit one word of data", "update head worklist pointer", etc.)

Asynchronous analysis

Pointer analysis detects interference with MMIO/DMA zones
Read/writes to these results in fixpoint iterations of async routines

Work in progress

Difficulty: I/O drivers are often very dirty

Comparison: Microsoft SLAM analyzes drivers, but apparently with ideal integers, does not handle many constructs, and totally ignores I/O controllers