

Case studies: Flight software verification and analysis of obfuscated binaries

Jan Midtgaard

Week 7, Abstract Interpretation

Aarhus University, Q4 - 2012

Two case studies:

- *Control-Flow Analysis of Function Calls and Returns by Abstract Interpretation*, Midtgaard and Jensen, ICFP'09 + IC'12
- *Abstract Debugging of Higher-Order Imperative Languages*, Bourdoncle, PLDI'93

Today

Two papers and then some:

- First paper: *A Static Analyzer for Large Safety-Critical Software*, Blanchet, Cousot, Cousot, Feret, Mauborgne, Miné, Monniaux, and Rival, PLDI'03
- A bit on compositional analysis
- Second paper: *Context-Sensitive Analysis of Obfuscated x86 Executables*, Lakhotia, Boccardo, Singh, and Manacero, PEPM'10 / HOSC'11
- Course retrospective and wrap-up

A Static Analyzer for Large Safety-Critical Software

[see PLDI'03 slides]

The ASTRÉE design refinement algorithm

1. Run static analysis with false alarms
2. Manually inspect cause of false alarm
3. Possible causes
 - Either rewrite abstract transfer function to strengthen it,
 - Refine a widening which is too coarse, or
 - Design a new abstract domain that can express the missed invariant
4. Wash, rinse, repeat

Compositional semantics and analysis (1/2)

ASTRÉE is based on *compositional* semantics and analysis: the semantics (or analysis) of a compound statement is a combination of the semantics (or analysis) of its parts.

For example:

$$\llbracket S_1 ; S_2 \rrbracket^\#(E^\#) = \llbracket S_2 \rrbracket \circ \llbracket S_1 \rrbracket^\#(E^\#)$$

$$\begin{aligned} \llbracket \text{if } c \text{ then } S_1 \text{ else } S_2 \rrbracket^\#(E^\#) &= \llbracket S_1 \rrbracket(\text{guard}^\#(E^\#, c)) \\ &\quad \sqcup^\# \llbracket S_2 \rrbracket(\text{guard}^\#(E^\#, \neg c)) \end{aligned}$$

$$\llbracket \text{while } c \text{ do } S \rrbracket^\#(E^\#) = \dots \text{lfp } F^\# \dots$$

$$\text{where } F^\#(E^{\#'}) = E^\# \sqcup^\# \llbracket S \rrbracket(\text{guard}^\#(E^{\#'}, c))$$

The classical transition system semantics and derived analyses we have studied **does not have this property**. 7 / 22

Compositional semantics and analysis (2/2)

The classical transition system semantics and derived analyses we have studied **does not have this property**.

Compositional **semantics** is preferable, because we can reason by structural induction.

Compositional **analysis** is preferable, because we avoid a global fixpoint computation:

- The analysis of sequential code is sequential
- The analysis of a loop involves a loop (namely fixpoint iteration)

In the end the resulting analysis is more efficient.

When combined with Bourdoncle's *minimal widening insight* the resulting analysis is also more precise.

Context-Sensitive Analysis of Obfuscated x86 Executables

[see PEPM'10 slides]

RIC Motivation

Suppose we want to represent the set of addresses $\{4000, 4004\}$?

Intervals would over-approximate:

$$\gamma([4000, 4004]) = \{4000, 4001, 4002, 4003, 4004\}$$

thereby losing track of the 4-byte alignment!

Simple congruences would be too imprecise:

$$\gamma(0 \bmod 4) = \{0, 4, \dots, 4000, 4004, \dots\}$$

RIC definition

RIC is short for *Reduced Interval Congruence*, which is a rather descriptive name:

Formally, the RIC domain is a triple:

$$RIC = \mathbb{N} \times \mathbb{Z} \times \mathbb{Z}$$

The meaning of an element is:

$$\gamma(s[lb, ub]) = \{z \mid lb \leq z \leq ub \wedge z = lb \bmod s\}$$

For example:

$$\gamma(2[1, 9]) = \{1, 3, 5, 7, 9\}$$

Abstractions for the toolbox

The paper presents two sequence abstractions, nicely formulated as Galois connections for the toolbox:

The k -CONTEXT ABSTRACTION cutting off after length k .

The l -CONTEXT ABSTRACTION collapsing loops of reoccurring elements.

Nicely packed and ready to take home...



Summary

Summary

Two case studies based on recent research articles:

- First paper: *A Static Analyzer for Large Safety-Critical Software*, Blanchet, Cousot, Cousot, Feret, Mauborgne, Miné, Monniaux, and Rival, PLDI'03
- A bit on compositional analysis
- Second paper: *Context-Sensitive Analysis of Obfuscated x86 Executables*, Lakhota, Boccardo, Singh, and Manacero, PEPM'10 / HOSC'11

Course retrospective and wrap-up

Recap of promises

[Abstract interpretation] is simply an alternative view — an eye opener to a new world.

It can be used to explain existing approaches and extend or strengthen them (e.g, using disjunctive completion, forward/backward analysis, . . .)

[You are now] in a position to make an informed opinion

It is not just an academic theory: it has been used to check/verify flight control software for both Airbus and Mars missions.

It [has been] bloody — there [was] mathematics — there [was] semantics

Learning outcomes and competences

The participants must at the end of the course be able to:

- *describe* and *explain* basic analyses in terms of classical abstract interpretation.
- *apply* and *reason* about Galois connections.
- *implement* abstract interpreters on the basis of the derived program analysis.

Suggestions for additions and changes are very welcome!