# Principles of Abstract Interpretation
# MIT press
# Ch. 1, Introduction

Patrick Cousot

pcousot.github.io

PrAbsInt@gmail.com  github.com/PrAbsInt/

These slides are available at
http://github.com/PrAbsInt/slides/slides/slides-01--introduction-PrAbsInt.pdf

Chapter 1

# Abstract interpretation and its main applications

# Software goes with bugs

Software is everywhere, so bugs are everywhere, and some bugs are catastrophic:

Ariane 5.01
(overflow)

Patriot
(float rounding)

Mars Orbiter
(unit error)

Mars Schiaparelli
(dimensioning
error)

Heartbleed
(missing
bounds check)

Most bugs are insidious and never show up: *e.g.* privacy in social networks

esamultimedia.esa.int/docs/esa-x-1819eng.pdf
en.wikipedia.org/wiki/MIM-104_Patriot
en.wikipedia.org/wiki/Mars_Climate_Orbiter
en.wikipedia.org/wiki/Schiaparelli_EDM
en.wikipedia.org/wiki/Heartbleed

# The poor state of software development (*e.g.* social networks)

- Software is enormous (tens of millions of lines)
- Built over very long periods of time (tens of years)
- Dozens of bugs are found daily
- Modified every day (bug corrections, enhancements, *etc.*)
- Programming team changes frequently (most original software designers have left)
- Leadership not held responsible[1]

→ Software is an untamed monster

[1] with rare exceptions NYT Jan. 23, 2019, NYT Dec. 13, 2013

# A possible remedy: formal methods

- Program verification: prove mathematically that the software satisfies requirements
  - → requirements can be specified by the programmer or a software analyst
  - → the programmer help is needed to state program properties
  - → theorem provers can help to do the mathematical proof
  - → verification works very well in the small
  - → *e.g.* Why3 used for the proof of C programs
- Static program analysis: is performed without actually executing programs
  - → requirements are often predefined (*e.g.* absence of runtime errors)
  - → program properties are inferred by the analyzer
  - → the analysis is fully automatic
  - → works well in the large
  - → *e.g.* Astrée for the analysis of runtime errors in C, Zoncolan for security analysis (data breaches), and Infer for concurrency analysis
- The main theoretical difference is on the level of abstraction

# Abstract interpretation [P. Cousot and R. Cousot, 1977, 1979]

- Mathematical foundations of formal methods
- Program semantics $\rightarrow$ $\begin{cases} \text{Program verification} \\ \text{Static program analysis} \end{cases}$

# Objective of the course

- To introduce the theory of abstract interpretation
- To discuss its application to
  - semantics
  - program verification
  - static program analysis

  for safety and security properties of programs.
- This first class: provide a simple introductory example

"Finding people who really know static analysis is very hard, you should tell your students that if they want a great job in a Silicon Valley company they should study abstract interpretation not JavaScript. Feel free to quote me on that ;-)"

Francesco Logozzo, Creator of the static analyzer Zoncolan for privacy analysis at Facebook

Companies developing static analyzers (in NYC): Amazon, Facebook, Google, IBM.


## Google Cloud DevOps is hiring!

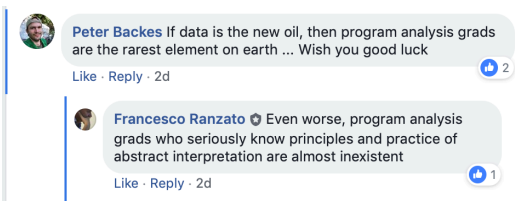
Published on January 8, 2019

Domagoj Babic | ✓ Following
Tech Lead & Manager @ Google
2 articles

15 0 1

**Job Description:**

The DevOps organization is looking for Software Engineers passionate about research and development of Google-scale deep program analysis tools, based on abstract interpretation. We are starting a new team that will develop cutting-edge deep Go static analysis tools and deploy them in production. We are looking for individuals who enjoy collaborative teamwork, thrive on computationally hard problems, are motivated by impact, deeply care about software correctness and security, and have a strong academic background in program analysis.



# Book and historical references

Cousot, Patrick (2021). *Principles of Abstract Interpretation*. 1st ed. MIT Press.

Cousot, Patrick and Radhia Cousot (1977). "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints". In: *POPL*. ACM, pp. 238–252.

— (1979). "Systematic Design of Program Analysis Frameworks". In: *POPL*. ACM Press, pp. 269–282.

# Home work

Read Ch. **1** "Abstract interpretation and its main applications" of

*Principles of Abstract Interpretation*
Patrick Cousot
MIT Press

# The End, Thank you