Abstract Interpretation: From Theory to Tools

Patrick Cousot

cims.nyu.edu/~pcousot/
pousot@cims.nyu.edu
Bugs everywhere!

Ariane 5.01 failure  
(overflow error)

Patriot failure  
(float rounding error)

Mars orbiter loss  
(unit error)

Russian Proton-M/DM-03 rocket  
carrying 3 Glonass-M satellites  
(unknown programming error :)

unsigned int payload = 18; /* Sequence number + random bytes */
unsigned int padding = 16; /* Use minimum padding */
/* Check if padding is too long, payload and padding 
must not exceed 2^14 - 3 = 16381 bytes in total.*/
OPENSSL_assert(payload + padding <= 16381);
/* Create HeartBeat message, we just use a sequence number 
- 1 byte to distinguish different messages and add 
the random stuff; */
buf = OPENSSL_malloc(1 + 2 + payload + padding);
p = buf;
/* Message Type */
*p++ = TLS1_HB_REQUEST;
/* Payload length (18 bytes here) */
s2n(payload, p);
/* Sequence number */
s2n(s->tlsext_hb_seq, p);
/* 16 random bytes */
RAND_pseudo_bytes(p, 16);
p += 16;
/* Random padding */
RAND_pseudo_bytes(p, padding);
ret = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buf, 3 + payload + padding);

Heartbleed  
(buffer overrun)
Bugs everywhere!

- Ariane 5.01 failure (overflow error)
- Patriot failure (float rounding error)
- Mars orbiter loss (unit error)
- Russian Proton-M/DM-03 rocket carrying 3 Glonass-M satellites (unknown programming error :)

- These are great proofs of the presence of bugs!

Heartbleed (buffer overrun)
On the limits of bug finding

• Giant software manufacturers can rely on gentle end-users to find myriads of bugs;

• But what about:

  can passengers really help?

• Is dynamic/static bug finding always enough?

• Proving the absence of bugs is much better!
Formal Methods
Formal Methods

- **Mathematical and engineering principles** applied to the specification, design, construction, verification, maintenance, and evolution of *very high quality* software

- Strongly promoted by **Harlan D. Mills** since the 70’s e.g.
  - ...
Main formal methods for verification

• **Objective**: prove automatically that a program does satisfy a specification given either explicitly or implicitly (e.g. absence of runtime errors)

• **Deductive methods**: use a theorem prover/proof assistant to check a user-provided proof argument

• Enumerative, symbolic, bounded, solver (e.g. Z3)-based, interpolation, statistical, etc **model-checking**: check the specification by enumerating *finitely many* possibilities

• **Abstract interpretation**: use approximation ideas to consider *infinitely many* possibilities
Fundamental limitations

• By Gödel’s **undecidability**, no perfect solution is and will ever be possible:
  
  • **Deductive methods**: the burden is on the end-user and the proofs are exponential in the size of programs
  
  • **Model-checking**: severe unsolved scalability problem
  
  • **Abstract interpretation**: may produce false alarms (but no false negative)
  
  • **Unsound methods** (Coverity, Klocwork, Purify, etc): no correctness guarantee at all.
The Evolution of Formal Methods
Change of Scale

• **1993:** IBM Flight Control. A HH60 helicopter avionics component was developed on schedule in three increments comprising 33 KLOC of JOVIAL [6]. A total of 79 corrections were required during **statistical certification** for an error rate of 2.3 errors per KLOC for verified software with no prior execution or debugging.

• **2013:** Astrée checks automatically the absence of any runtime error in the control/command software of the A380 and A400M by **abstract interpretation** i.e. > 1000 KLOC of C


Proliferation

WCET
Axiomatic semantics
Confidentiality analysis
Program synthesis
Grammar analysis
Statistical model-checking
Invariance proof
Probabilistic verification
Parsing

Security protocol verification
Dataflow analysis
Model checking
Denotational semantics
Obfuscation
Dependence analysis
CEGAR
Program transformation

Systems biology analysis
Database query
Abstraction refinement
Type inference
Separation logic
Termination proof

Type inference
Shape analysis
Malware detection
Code refactoring

Operational semantics
Abstraction refinement
Type inference
Separation logic
Termination proof

Confidentiality analysis
Program synthesis
Grammar analysis
Statistical model-checking
Invariance proof
Probabilistic verification
Parsing

Security protocol verification
Dataflow analysis
Model checking
Denotational semantics
Obfuscation
Dependence analysis
CEGAR
Program transformation

Systems biology analysis
Database query
Abstraction refinement
Type inference
Separation logic
Termination proof

Type inference
Shape analysis
Malware detection
Code refactoring

Operational semantics
Abstraction refinement
Type inference
Separation logic
Termination proof

Confidentiality analysis
Program synthesis
Grammar analysis
Statistical model-checking
Invariance proof
Probabilistic verification
Parsing

Security protocol verification
Dataflow analysis
Model checking
Denotational semantics
Obfuscation
Dependence analysis
CEGAR
Program transformation

Systems biology analysis
Database query
Abstraction refinement
Type inference
Separation logic
Termination proof

Type inference
Shape analysis
Malware detection
Code refactoring

Operational semantics
Abstraction refinement
Type inference
Separation logic
Termination proof

Confidentiality analysis
Program synthesis
Grammar analysis
Statistical model-checking
Invariance proof
Probabilistic verification
Parsing

Security protocol verification
Dataflow analysis
Model checking
Denotational semantics
Obfuscation
Dependence analysis
CEGAR
Program transformation

Systems biology analysis
Database query
Abstraction refinement
Type inference
Separation logic
Termination proof

Type inference
Shape analysis
Malware detection
Code refactoring

Operational semantics
Abstraction refinement
Type inference
Separation logic
Termination proof

Confidentiality analysis
Program synthesis
Grammar analysis
Statistical model-checking
Invariance proof
Probabilistic verification
Parsing

Security protocol verification
Dataflow analysis
Model checking
Denotational semantics
Obfuscation
Dependence analysis
CEGAR
Program transformation

Systems biology analysis
Database query
Abstraction refinement
Type inference
Separation logic
Termination proof

Type inference
Shape analysis
Malware detection
Code refactoring

Operational semantics
Abstraction refinement
Type inference
Separation logic
Termination proof

Confidentiality analysis
Program synthesis
Grammar analysis
Statistical model-checking
Invariance proof
Probabilistic verification
Parsing

Security protocol verification
Dataflow analysis
Model checking
Denotational semantics
Obfuscation
Dependence analysis
CEGAR
Program transformation

Systems biology analysis
Database query
Abstraction refinement
Type inference
Separation logic
Termination proof

Type inference
Shape analysis
Malware detection
Code refactoring

Operational semantics
Abstraction refinement
Type inference
Separation logic
Termination proof

Confidentiality analysis
Program synthesis
Grammar analysis
Statistical model-checking
Invariance proof
Probabilistic verification
Parsing

Security protocol verification
Dataflow analysis
Model checking
Denotational semantics
Obfuscation
Dependence analysis
CEGAR
Program transformation

Systems biology analysis
Database query
Abstraction refinement
Type inference
Separation logic
Termination proof

Type inference
Shape analysis
Malware detection
Code refactoring

Operational semantics
Abstraction refinement
Type inference
Separation logic
Termination proof

Confidentiality analysis
Program synthesis
Grammar analysis
Statistical model-checking
Invariance proof
Probabilistic verification
Parsing

Security protocol verification
Dataflow analysis
Model checking
Denotational semantics
Obfuscation
Dependence analysis
CEGAR
Program transformation

Systems biology analysis
Database query
Abstraction refinement
Type inference
Separation logic
Termination proof

Type inference
Shape analysis
Malware detection
Code refactoring

Operational semantics
Abstraction refinement
Type inference
Separation logic
Termination proof

Confidentiality analysis
Program synthesis
Grammar analysis
Statistical model-checking
Invariance proof
Probabilistic verification
Parsing

Security protocol verification
Dataflow analysis
Model checking
Denotational semantics
Obfuscation
Dependence analysis
CEGAR
Program transformation

Systems biology analysis
Database query
Abstraction refinement
Type inference
Separation logic
Termination proof

Type inference
Shape analysis
Malware detection
Code refactoring

Operational semantics
Abstraction refinement
Type inference
Separation logic
Termination proof

Confidentiality analysis
Program synthesis
Grammar analysis
Statistical model-checking
Invariance proof
Probabilistic verification
Parsing

Security protocol verification
Dataflow analysis
Model checking
Denotational semantics
Obfuscation
Dependence analysis
CEGAR
Program transformation

Systems biology analysis
Database query
Abstraction refinement
Type inference
Separation logic
Termination proof

Type inference
Shape analysis
Malware detection
Code refactoring

Operational semantics
Abstraction refinement
Type inference
Separation logic
Termination proof
The *Theory of Abstract Interpretation:* Unifies Formal Methods
The need for a unified account of formal methods

- Operational semantics
- Abstraction refinement
- Type inference
- Separation logic
- Termination proof
- Shape analysis
- Malware detection
- Code refactoring

- Security protocol verification
- Systems biology analysis
- Database query
- Dependence analysis
- Dependence analysis

- Denotational semantics
- Model checking
- Database query
- CEGAR
- Program transformation
- Abstract model checking

- Partial evaluation
- Obfuscation
- Dataflow analysis
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Axiomatic semantics
- Dataflow analysis
- Partial evaluation
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Confidentiality analysis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis

- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof

- WCET
- Security protocol verification
- Systems biology analysis
- Database query
- Dependence analysis
- Dependence analysis

- Partial evaluation
- Obfuscation
- Dataflow analysis
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Axiomatic semantics
- Dataflow analysis
- Partial evaluation
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Confidentiality analysis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis

- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof

- WCET
- Security protocol verification
- Systems biology analysis
- Database query
- Dependence analysis
- Dependence analysis

- Partial evaluation
- Obfuscation
- Dataflow analysis
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Axiomatic semantics
- Dataflow analysis
- Partial evaluation
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Confidentiality analysis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis

- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof

- WCET
- Security protocol verification
- Systems biology analysis
- Database query
- Dependence analysis
- Dependence analysis

- Partial evaluation
- Obfuscation
- Dataflow analysis
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Axiomatic semantics
- Dataflow analysis
- Partial evaluation
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Confidentiality analysis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis

- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof

- WCET
- Security protocol verification
- Systems biology analysis
- Database query
- Dependence analysis
- Dependence analysis

- Partial evaluation
- Obfuscation
- Dataflow analysis
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Axiomatic semantics
- Dataflow analysis
- Partial evaluation
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Confidentiality analysis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis

- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof

- WCET
- Security protocol verification
- Systems biology analysis
- Database query
- Dependence analysis
- Dependence analysis

- Partial evaluation
- Obfuscation
- Dataflow analysis
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Axiomatic semantics
- Dataflow analysis
- Partial evaluation
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Confidentiality analysis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis

- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof

- WCET
- Security protocol verification
- Systems biology analysis
- Database query
- Dependence analysis
- Dependence analysis

- Partial evaluation
- Obfuscation
- Dataflow analysis
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Axiomatic semantics
- Dataflow analysis
- Partial evaluation
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Confidentiality analysis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis

- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof

- WCET
- Security protocol verification
- Systems biology analysis
- Database query
- Dependence analysis
- Dependence analysis

- Partial evaluation
- Obfuscation
- Dataflow analysis
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Axiomatic semantics
- Dataflow analysis
- Partial evaluation
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Confidentiality analysis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis

- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof

- WCET
- Security protocol verification
- Systems biology analysis
- Database query
- Dependence analysis
- Dependence analysis

- Partial evaluation
- Obfuscation
- Dataflow analysis
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Axiomatic semantics
- Dataflow analysis
- Partial evaluation
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Confidentiality analysis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis

- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof

- WCET
- Security protocol verification
- Systems biology analysis
- Database query
- Dependence analysis
- Dependence analysis

- Partial evaluation
- Obfuscation
- Dataflow analysis
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Axiomatic semantics
- Dataflow analysis
- Partial evaluation
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Confidentiality analysis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis

- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof

- WCET
- Security protocol verification
- Systems biology analysis
- Database query
- Dependence analysis
- Dependence analysis

- Partial evaluation
- Obfuscation
- Dataflow analysis
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Axiomatic semantics
- Dataflow analysis
- Partial evaluation
- Trace semantics
- Denotational semantics
- CEGAR
- Partial evaluation

- Confidentiality analysis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis
- Program synthesis

- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof
- Statistical model-checking
- Invariance proof

Underlying unity of formal methods

Abstract interpretation

- WCET
- Axiomatic semantics
- Confidentiality analysis
- Program synthesis
- Grammar analysis
- Statistical model-checking
- Invariance proof
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Obfuscation
- Denotational semantics
- Trace semantics
- Effect systems
- Partial evaluation
- Symbolic execution
- Quantum entanglement detection
- Type theory
- Steganography
- Probabilistic verification
- Parsing
Principle of Abstract Interpretation
What is abstraction in AI?

Concrete universe of discourse
What is abstraction in AI?

Concrete universe of discourse

Elements
What is abstraction in AI?

Concrete universe of discourse

Elements

Properties
What is abstraction in AI?

Concrete universe of discourse

Elements

Properties

Abstract universe of properties
What is abstraction in AI?

Concrete universe of discourse

Elements

Properties

Abstract universe of properties

Abstract properties
What is abstraction in AI?

Concrete universe of discourse

Elements

Properties

Abstract universe of properties

Abstract properties

\( \alpha \)

\( \gamma \)
What is abstraction in AI?

Concrete universe of discourse

Elements

Properties

Abstract universe of properties

Abstract properties

Inclusion

\( \alpha \subseteq \gamma \)
What is abstraction in AI?

Concrete universe of discourse

Elements

Properties

Abstract universe of properties

Abstract implication

Inclusion

\( \subseteq \)

\( \alpha \)

\( \gamma \)
What is abstraction in AI?

Concrete universe of discourse

Elements

Properties

Inclusion

Abstract universe of properties

Abstract properties

Abstract implication

Provable abstract properties are true in the concrete
We use a Galois connection \((C, \preceq) \rightleftharpoons (A, \sqsubseteq)\) as the absence of runtime errors (used to validate both the user-provided contracts as well as Descending chain conditions, a fixpoint acceleration operating at different levels of abstraction. The underlying tools computing an approximation of a greatest lower bound (lb), false, true, \(\land, \lor, \neg\)) are guaranteed to be definitely invalid or will not terminate. The difference is that this sufficient precondition might exclude valid methods. We think that method extraction should come with numerous (and self-evident) complaints from end-users of extracted methods. We think that this is overkill and that the problem is to require the user to provide the precondition exactly the length of the array holds.

The precondition abstract domain \((A[\overline{v}], \sqsubseteq)\) is an abstract domain expressing properties of the variables \(\overline{v}\) where the partial order \(\sqsubseteq\) aborts logical implication. The meaning of an abstract property \(P \in A[\overline{v}]\) is a concrete property \(\gamma_1(P) \in P[\overline{v}]\) where the concretization \(\gamma_1(A[\overline{v}], \sqsubseteq) \rightarrow (P[\overline{v}], \sqsubseteq)\) is increasing (i.e., \(P \sqsubseteq P'\) implies \(\gamma_1(P) \rightarrow \gamma_1(P')\)).
A very informal introduction to abstract interpretation

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282
1) Define the programming language semantics

Formalize the concrete **executions** of programs (e.g. transition system)

Trajectory in state space

Space/time trajectory
II) Define the program properties of interest

Formalize what you are interested to know about program behaviors

We are interested in the set of possible trajectories
III) Define which specification must be checked

Formalize what you are interested to prove about program behaviors

Forbidzen zone

No trajectory should hit the forbidden zone
IV) Choose the appropriate abstraction

Abstract away all information on program behaviors irrelevant to the proof

Abstraction by geometric forms (rectangles, polyhedra, ellipsoids, abstraction by parts, etc)
V) Mechanically verify in the abstract

The proof is fully **automatic**

Forbidden zone

Abstraction of the trajectories

Provable abstract properties are true in the concrete
Soundness of the abstract verification

Never forget any possible case so the **abstract proof is correct in the concrete**
Unsound validation: testing

Try a few cases

Forbidden zone

Error !!!

Test of a few trajectories
Unsound validation: bounded model-checking

Simulate the beginning of all executions

Forbidden zone

Bounded model-checking

Error !!!

Possible trajectories
Unsound validation: static analysis

Many static analysis tools are **unsound** (e.g. Coverity, etc.) so inconclusive
Incompleteness

When abstract proofs may fail while concrete proofs would succeed

By soundness an alarm must be raised for this overapproximation!
True error

The abstract alarm may correspond to a concrete error
False alarm

The abstract alarm may correspond to no concrete error (false negative)

The only solution is to refine the analysis to take more properties into account (e.g. specifically for a domain of application)!
Combination of abstractions in Astrée

II.P. Combination of abstract domains

Abstract interpretation based tools usually use several different abstract domains since the design of a complex one is best decomposed into a combination of simpler abstract domains. Here are a few abstract domain examples used in the Astrée static analyzer:

- Collecting semantics: partial traces
- Intervals: \( x \in [a, b] \)
- Simple congruences: \( x \equiv a[b] \)
- Octagons: \( \pm x \pm y \leq a \)
- Ellipses: \( x^2 + by^2 - axy \leq d \)
- Exponentials: \( -a^{bt} \leq y(t) \leq a^{bt} \)

Such abstract domains are described in more detail in Sections III–III–II. The following classic abstract domains, however, are not used in Astrée because they are either too imprecise, not scalable, or difficult to implement correctly (for instance, soundness may be an issue in the event of floating-point rounding or out of scope for determining program properties which are usually of no interest to prove the specification):

- Polyhedra
- Signs
- Linear congruences

Because abstract domains do not use a uniform machine representation of the information they manipulate, combining them is not completely trivial. The conjunction of abstract program properties has to be performed, ideally, by a reduced product for Galois connection abstractions. In absence of a Galois connection or for performance reasons, the conjunction is performed using an easily computable but not optimal overapproximation of this combination of abstract domains.

Assume that we have designed several abstract domains and computed \( \text{lfp } F_1 \cup \cdots \cup F_n \) in these abstract domains \( D_1 \cup \cdots \cup D_n \) relative to a collecting semantics \( C \). The combination of these analyses is sound as \( C \uparrow \text{lfp } F_1 \uparrow \cdots \uparrow \text{lfp } F_n \). However, only combining the analysis results is not very precise as it does not permit analyses to improve each other during the computation.
Examples of abstract interpretation-based program verification tools
Example 1: Astrée
Astrée

- Commercially available: www.absint.com/astree/

- Effectively used in production to qualify truly large and complex software in transportation, communications, medicine, etc.

Example of domain-specific abstraction: ellipses

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 [??????, ??????] */
}

void main () { X = 0.2 * X + 5; INIT = TRUE;
    while (1) {
        X = 0.9 * X + 35; /* simulated filter input */
        filter (); INIT = FALSE; }
}
Abstract interpretation

• Abstract interpretation is the only formal method able to automatically infer program properties

• All others can only check your assertions
Example of domain-specific abstraction: ellipses

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 [??????, ??????] */
}

void main () { X = 0.2 * X + 5; INIT = TRUE;
    while (1) {
        X = 0.9 * X + 35; /* simulated filter input */
        filter (); INIT = FALSE; }
}
Example of domain-specific abstraction: ellipses

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 [-1418.3753, 1418.3753] */
}

void main () { X = 0.2 * X + 5; INIT = TRUE;
    while (1) {
        X = 0.9 * X + 35; /* simulated filter input */
        filter (); INIT = FALSE; }
}
Example II: cccheck
Code Contract Static Checker (cccheck)

- Available within MS Visual Studio

Manuel Fähndrich, Francesco Logozzo: Static Contract Checking with Abstract Interpretation. FoVeOOS 2010: 10-30
Comments on screenshot (courtesy Francesco Logozzo)

- A screenshot from Clousot/cccheck on the classic binary search.
- The screenshot shows from left to right and top to bottom
  1. C# code + CodeContracts with a buggy BinarySearch
  2. cccheck integration in VS (right pane with all the options integrated in the VS project system)
  3. cccheck messages in the VS error list
- The features of cccheck that it shows are:
  1. basic abstract interpretation:
     a. the loop invariant to prove the array access correct and that the arithmetic operation may overflow is inferred fully automatically
     b. different from deductive methods as e.g. ESC/Java or Boogie or Dafny where the loop invariant must be provided by the end-user
  2. inference of necessary preconditions:
     a. Clousot finds that array may be null (message 3)
     b. Clousot suggests and propagates a necessary precondition invariant (message 1)
  3. array analysis (+ disjunctive reasoning):
     a. to prove the postcondition one must infer properties of the content of the array
     b. please note that the postcondition is true even if there is no precondition requiring the array to be sorted.
  4. verified code repairs:
     a. from the inferred loop invariant does not follow that index computation does not overflow
     b. suggest a code fix for it (message 2)
Conclusion
To explore abstract interpretation...

- A good starting point:


Conclusion

• 40 years after Harlan D. Mills pioneer ideas, abstract interpretation-based formal methods have made considerable progress both in theory and practice

• May become indispensable as
  • safety and security become central to computer science
  • programmers are held responsible for their errors
  • machines hence programming becomes more and more complicated (if not intractable, e.g. parallelism, cloud, etc)
The End, Thank You