Introduction to abstract interpretation

Marc Chevalier

ENS de Lyon – ENS – INRIA

13 dec. 2016
What? Why?
What is abstract interpretation?
Why are we doing this?

Ok, but how?
An example
Correct examples
Question

Being more general
Abstraction: a formal framework
It’s great! But not so much
So... ?
Other abstract domains

In real world
Existing tools
Proved software
The safeword is

banana
What? Why?

What is abstract interpretation?
Why are we doing this?

Ok, but how?

An example
Correct examples
Question

Being more general

Abstraction: a formal framework
It’s great! But not so much
So... ?
Other abstract domains

In real world

Existing tools
Proved software
What is abstract interpretation?

A way to PROVE programs
What is abstract interpretation?

A way to PROVE programs
- No undefined behaviour
What is abstract interpretation?

A way to PROVE programs

- No undefined behaviour
- Specification
What is abstract interpretation?

A way to PROVE programs
- No undefined behaviour
- Specification
- Execution path/time
What is abstract interpretation?

A way to PROVE programs

- No undefined behaviour
- Specification
- Execution path/time
- ... any other property you can think of (which we can express in term of trace semantic).
Why are we doing this?

Figure 1: Ariane V, 4th June 1996
Why are we doing this?

Figure 1: Ariane V, 40s later
Why are we doing this?

There are other ways to convince oneself nothing bad happens:

- tests
- code style
- high-level language (with exception handling for instance)

But not enough: Ariane V (A lot of tests, good code style, Ada).
Why are we doing this?

There are other ways to convince oneself nothing bad happens

- tests
- code style
- high-level language (with exception handling for instance)

But not enough: Ariane V (A lot of tests, good code style, Ada).

We don’t want the slightest chance of error in critical code.
Why are we doing this?

Bugs have various annoying consequences:
Why are we doing this?

Bugs have various annoying consequences:

- deaths: Patriot MIM-104 (28), Toyota(≥89), radiotherapy machines
Why are we doing this?

Bugs have various annoying consequences:

- deaths: Patriot MIM-104 (28), Toyota(≥89), radiotherapy machines
- privacy: Hearthbleed
Why are we doing this?

Bugs have various annoying consequences:

- deaths: Patriot MIM-104 (28), Toyota(≥89), radiotherapy machines
- privacy: Heartbleed
- a LOT of money: 370 000 000 USD for Ariane, 59.5 billion USD annually in US (NIST)
What? Why?
What is abstract interpretation?
Why are we doing this?

Ok, but how?
An example
Correct examples
Question

Being more general
Abstraction: a formal framework
It’s great! But not so much
So… ?
Other abstract domains

In real world
Existing tools
Proved software
An example

```
int f(int x)
{
    x = abs(x);
    x = x + 1;
    return 1/x;
}
```
An example

```c
int f(int x)
{
    x = abs(x);
    x = x + 1;
    return 1/x;
}
```

// $x_1 \in [-2^{31}; 2^{31} - 1]$
An example

```c
int f(int x) {
    // x₁ ∈ [−2^{31}; 2^{31} − 1]
    x = abs(x); // x₃ ∈ [0; 2^{31} − 1] ∨ x₂ = −2^{31}
    x = x + 1;
    return 1/x;
}
```
An example

```c
int f(int x)
{
    // x_1 \in [-2^{31}; 2^{31} - 1]
    x = abs(x); // x_3 \in [0; 2^{31} - 1] \lor x_2 = -2^{31}
    x = x + 1; // x_4 \in [1; 2^{31} - 1] \lor x_3 = 2^{31} - 1
    return 1/x;
}
```
An example

```c
int f(int x)
{
    // x1 ∈ [−2^31; 2^31 − 1]
    x = abs(x); // x3 ∈ [0; 2^31 − 1] ∨ x2 = −2^31
    // x4 ∈ [1; 2^31 − 1] ∨ x3 = 2^31 − 1
    x = x + 1;
    return 1/x; // 0 ∉ [1; 2^31 − 1] ⇒ OK!
}
```
Correct examples

With contract:

```c
int f(int x)
{
    x = abs(x);  // x ∈ [0; 2^{31} - 2]
    x = x + 1;   // x ∈ [1; 2^{31} - 1]
    return 1/x;  // 0 ∉ [1; 2^{31} - 1] ⇒ OK !
}
```
Correct examples

Without contract:

```c
int f(int x)
{
    if(-(1<<31)+2 > x || x > 1<<31-2)
        return 0;
    // x ∈ [−2^{31} − 2; 2^{31} − 2]
    x = abs(x); // x ∈ [0; 2^{31} − 2]
    x = x + 1; // x ∈ [1; 2^{31} − 1]
    return 1/x; // 0 ∉ [1; 2^{31} − 1] ⇒ OK !
}```
Question

These analyses were flow-sensitive.

Why do we use flow-insensitive analyses?
What? Why?
What is abstract interpretation?
Why are we doing this?

Ok, but how?
An example
Correct examples
Question

Being more general
Abstraction: a formal framework
It’s great! But not so much
So... ?
Other abstract domains

In real world
Existing tools
Proved software
Abstraction: a formal framework

- $\mathcal{A}$: arithmetic set
- $\mathcal{V}$: variable set
- $\mathcal{M} = \mathcal{V} \rightarrow \mathcal{A}$: memory environment
- $D^\#$: abstract domain
- $\gamma : D^\# \rightarrow \mathcal{P}(\mathcal{M})$: concretization
Abstraction: a formal framework

- $\mathbb{A}$: arithmetic set: $[-2^{31}, -2^{-31} - 1]$
- $\mathbb{V}$: variable set
- $\mathbb{M} = \mathbb{V} \rightarrow \mathbb{A}$: memory environment
- $D^\#$: abstract domain
- $\gamma : D^\# \rightarrow \mathcal{P}(\mathbb{M})$: concretization
Abstraction: a formal framework

- $\mathbb{A}$: arithmetic set: $[-2^{31}, 2^{31} - 1]$
- $\mathbb{V}$: variable set: $\{x\}$
- $\mathbb{M} = \mathbb{V} \rightarrow \mathbb{A}$: memory environment
- $D^\#: \text{abstract domain}$
- $\gamma: D^\# \rightarrow \mathcal{P}(\mathbb{M})$: concretization
Abstraction: a formal framework

- \( \mathbb{A} \): arithmetic set: \([-2^{31}, -2^{-31} - 1]\)
- \( \mathbb{V} \): variable set: \(\{x\}\)
- \( \mathbb{M} = \mathbb{V} \rightarrow \mathbb{A} \): memory environment: \(x: 42\)
- \( D^\# \): abstract domain
- \( \gamma : D^\# \rightarrow \mathcal{P}(\mathbb{M}) \): concretization
Abstraction: a formal framework

- $\mathbb{A}$: arithmetic set: $[-2^{31}, -2^{-31} - 1]$
- $\mathbb{V}$: variable set: $\{x\}$
- $\mathbb{M} = \mathbb{V} \rightarrow \mathbb{A}$: memory environment: $x: 42$
- $D^\#$: abstract domain:
  \[ \{”v \in [a, b]” \mid v \in \mathbb{V}, (a, b) \in \mathbb{A}^2\} \]
- $\gamma: D^\# \rightarrow \mathcal{P}(\mathbb{M})$: concretization
Abstraction: a formal framework

- $\mathbb{A}$: arithmetic set: $[-2^{31}, -2^{-31} - 1]$
- $\mathbb{V}$: variable set: $\{x\}$
- $\mathbb{M} = \mathbb{V} \rightarrow \mathbb{A}$: memory environment: $x : 42$
- $D^\#$: abstract domain:
  $\{"v \in [a, b]" \mid v \in \mathbb{V}, (a, b) \in \mathbb{A}^2\}$
- $\gamma : D^\# \rightarrow \mathcal{P}(\mathbb{M})$: concretization:
  $\"v \in [a, b]\" \mapsto \{v : c \mid c \in [a, b]\}$
Abstraction: a formal framework

- $A$: arithmetic set: $[-2^{31}, -2^{-31} - 1]$
- $V$: variable set: $\{x\}$
- $M = V \rightarrow A$: memory environment: $x: 42$
- $D^\#: $ abstract domain: $\{"v \in [a, b]" \mid v \in V, (a, b) \in A^2\}$
- $\gamma: D^\# \rightarrow \mathcal{P}(\mathcal{M})$: concretization: $"v \in [a, b]" \mapsto \{v : c \mid c \in [a, b]\}$

Sound: if $\gamma$ is an over-approximation, catch all possible behaviours (and maybe more).
It’s great! But not so much

**Theorem 1.**

A sound abstraction will find every errors.

**Theorem 2.**

A sound abstraction will find problems where there are none.
It’s great! But not so much

```c
/*@ requires -10 <= x <= 10;
   ensures \result >= 0;

int g(int x)
{
    // x ∈ [−10; 10]
    x = x*x  // x ∈ [−100; 100]
    return x;  // [−100; 100] ∉ [0; +∞]⇒ ERROR !
}
```

Not too disappointed?
**It’s great! But not so much**

```c
/*@ requires -10 <= x <= 10;
   ensures \result >= 0;
*/
int g(int x)
{
    // x ∈ [-10; 10]
    x = x*x   // x ∈ [-100; 100]
    return x; // [-100;100] ⊆ [0;+∞] ⇒ ERROR!
}
```

Not too disappointed?

What happens? The abstract domain cannot understand the relation between \(x\) (lhs) and \(x\) (rhs).
So...?

There are several solutions:

More and more complex abstract domains (relational, memory domain, trace abstraction, partitioning etc.): **fully automatic**, a lot of development
Introduction to abstract interpretation

Marc Chevalier

What? Why?
Ok, but how?
Being more general
Abstraction: a formal framework
It’s great! But not so much
So...?
Other abstract domains
In real world

So...?

Human driven analysis: light development, need human intervention for each analysis

```c
/*@ requires -10 <= x <= 10;
  ensures \result >= 0;
 */

int g(int x)
{
  //@ assert x < 0 || x >= 0;
  // x ∈ [−10, −1]
  // x ∈ [0, 10]
  x = x*x   // x ∈ [1, 100]
    // x ∈ [0, 100]
  return x; // x ∈ [1, 100] ∪ [0, 100] = [0, 100]
    // [0, 100] ⊆ [0, +∞] ⇒ OK
}
```
Other abstract domains

- Polytope: $\sum a_i x_i \leq c_i$
- Modulo: $x_i \equiv c_i [n_i]$
- Partitioning: $x \geq 0 \Rightarrow \ldots \land x < 0 \Rightarrow \ldots$
- Memory abstraction (struct, pointer, separation logic)
- Symbolic: $x - x = 0$
- Product domain: combination of other domains
What? Why?
What is abstract interpretation?
Why are we doing this?

Ok, but how?
An example
Correct examples
Question

Being more general
Abstraction: a formal framework
It’s great! But not so much
So…?
Other abstract domains

In real world
Existing tools
Proved software
Existing tools

Open-source:
- Frama-C (CEA LIST, INRIA, TrustInSoft): C, C++ (in progress)
- Code Contract Static Checker (Microsoft Research): .NET

Proprietary:
- Astrée (CNRS, ENS, INRIA, AbsInt): embedded C, inline x86 soon
- Polyspace (MathWorks): Ada, C, C++
- aiT (AbsInt): Worst-case execution time for binary
- Sparrow (Seoul National University): C
- Julia (University of Verona): Java and Android
Proved software

- Nuclear plant
- Airbus (in progress) (my current work)
- Automatic subway
- Telecommunications
- And a lot of super secret and critic things (past work)
- PolarSSL
- libiconv (past work)
Thanks for your attention

Any questions?

marc.chevalier@ens-lyon.fr