# Program Analysis

## Metrics, Soundness, Lattices, AI

Yegor Bugayenko

Lecture #6 out of 10
90 minutes

All videos are in this YouTube playlist.

Chapter #1:

# Basics

# Syntactic & Semantic Properties

Semantic property can be completely defined with respect to the set of executions of a program, while a syntactic property can be decided directly based on the program text.

```
if (x) { printf("大家好"); }
```

Which properties are dynamic?

- A program may print a text to the console
- A program may call `printf()` C library function
- A program prints to the console
- A program consists of one line of code

# Rice's Theorem

Rice's theorem states that all non-trivial semantic properties of programs are undecidable.

A property is non-trivial if it is neither true for every partial computable function, nor false for every partial computable function.

Halting problem is the problem of determining, from 1) a description of an arbitrary computer program and 2) an input, whether the program will finish running, or continue to run forever. A general algorithm to solve the halting problem for all possible program–input pairs **cannot exist**.

# Non-trivial Properties

Examples of a non-trivial properties:

- A program exits
- A program prints “Hello”
- A program finishes in less than 5 seconds
- A program dies with “Segmentation Fault”
- A program prints user password to the console

Suggest a few properties.

# Static Analysis

Consider two C++ programs given to a static analyzer (e.g. Clang Tidy):

```
int f() {
  int x = 0;
  return 42 / x;
}
```

```
int f(int x) {
  return 42 / x;
}
```

Expected answers from Clang Tidy:

**Yes! :)**

**No :(**

# Style Checking

Consider two C++ programs given to a style checker (e.g. cpplint):

```
int f (int x)
{
  return 42 / x;
}
```

```
int foo(int x) {
  return 42 / x;
}
```

Expected answers from cpplint:

```
Extra space before ( in
function call ; { should
almost always be at the end
of the previous line
```

**No :(**

# Dynamic Analysis

Consider this C++ program (Clang Tidy finds no issues) given to a dynamic analyzer (AddressSanitizer):

```
int foo(int i) {
  int a[5];
  return a[i];
}
int main() {
  return foo(6);
}
```

```
$ gcc -fsanitize=address -g a.cpp
$ ./a.out
```

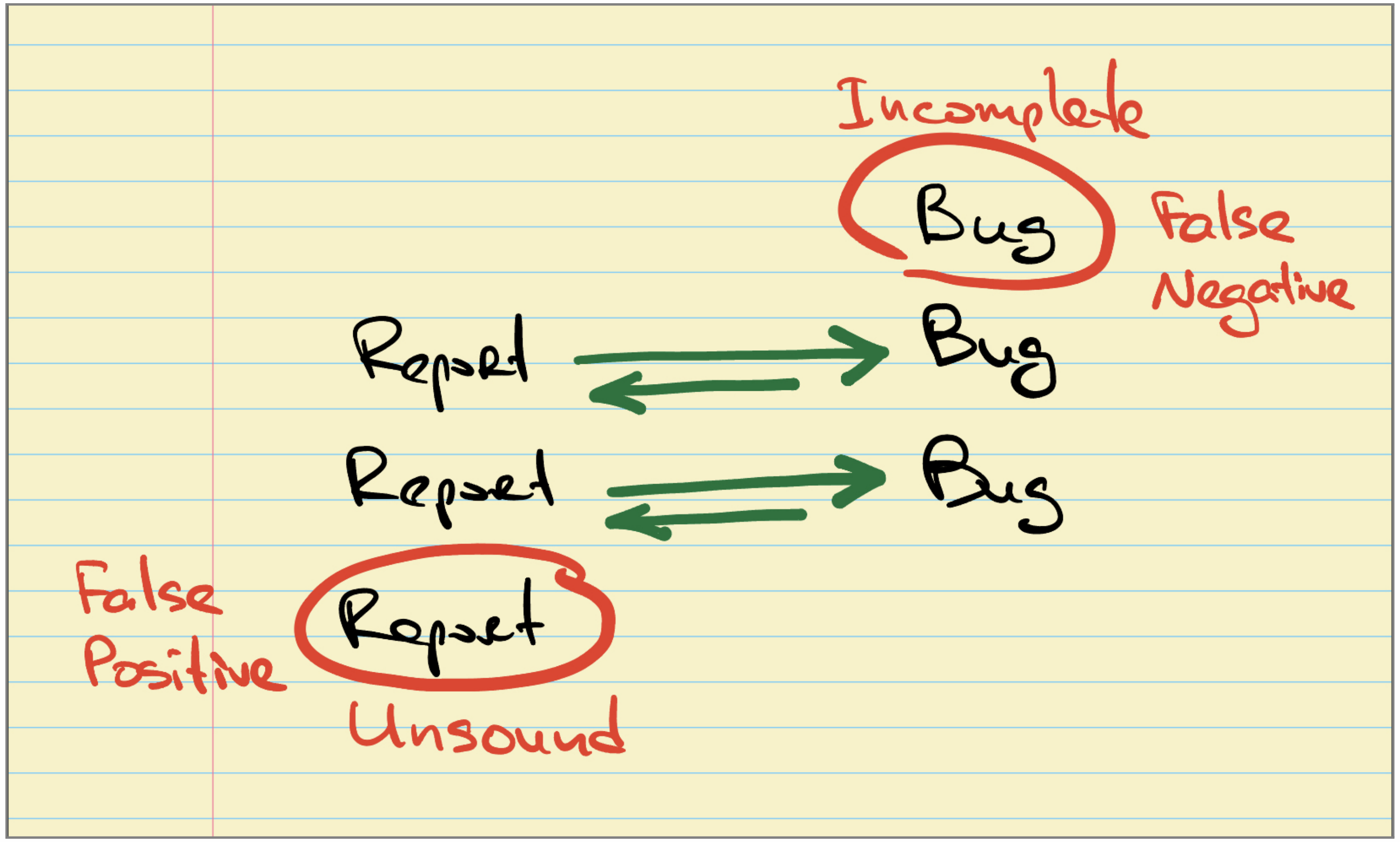
```
=================================================================
==76375==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x00016babf0d8
READ of size 4 at 0x00016babf0d8 thread T0
    #0 0x104343e54 in foo(int) a.cpp:9
    #1 0x104343f38 in main a.cpp:12
    #2 0x1a07c7e4c  (<unknown module>)

Address 0x00016babf0d8 is located in stack of thread T0 at offset 56 in frame
    #0 0x104343cf0 in foo(int) a.cpp:7

  This frame has 1 object(s):
    [32, 52) 'a' (line 8) <== Memory access at offset 56 overflows this variable
```

Dynamic analysis == testing.

Chapter #2:

# Quality of Analysis

# Sound & Complete



# Precision & Recall

Precision is the fraction of relevant instances among the retrieved instances (100% precision means soundness).

Recall is the fraction of relevant instances that were retrieved (100% recall means completeness).

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN} \qquad \text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{F1} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

# Experiment

Say, we give a few programs to a static analyzer:

| a | b | c | d | e | f | g | h |
|---|---|---|---|---|---|---|---|
| Yes | Yes | No | Yes | No | Yes | Yes | No |

$$TP = \_\_ \qquad FP = \_\_ \qquad TN = \_\_ \qquad FN = \_\_$$

$$\text{Precision} = \frac{TP}{TP + FP} = \_\_\_\_\_\_\_\_ \qquad \text{Recall} = \frac{TP}{TP + FN} = \_\_\_\_\_\_\_\_$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \_\_\_\_\_\_ \qquad \text{F1} = \frac{2 \times TP}{2 \times TP + FP + FN} = \_\_\_\_\_\_$$

# Flip of Terminology

Soundness and Completeness: With Precision by Prof. Bertrand Meyer, in Blog@CACM: “It is very easy to obtain soundness if we forsake completeness: reject every case.”

Chapter #3:

# Lattice

# Total Order

Total order is a binary relation $\leq$ (strict total order is $<$).

Lineary ordered set (loset) is a set equipped with a total order.

Which of them are losets?:

$$\{1, -5, 2, 0, 42\}$$

$$\{3, 5, -9, 5, 12\}$$

$$\{3, 5, \texttt{"Hello"}, 12, 5.0\}$$

$$\{x, y, z\}$$

$$\varnothing$$

# Partially Ordered Set

Partial order is total order but only between some elements.

Partially ordered set (poset) is a set equipped with a partial order.

Which of them are posets?:

$\{1, \texttt{"apple"}, 2, -7, \texttt{"orange"}\}$

$\{3, 5, -9, 5, 12\}$

$\{0, 1, 2, 3, \ldots\}$

$\{x, y, z\}$

$\varnothing$

# Lattice

Lattice is a poset where each two elements $(x, y)$ have least upper bound (join operator $x \sqcup y$) and greatest lower bound (meet operator $x \sqcap y$).

$\{42, 2, 13\}$

$$\begin{tikzcd} 42 \\ 13 \arrow[u] \\ 2 \arrow[u] \end{tikzcd}$$

$42 \sqcup 2 = \ldots$

$\{\mathtt{A}, 7, 19, \mathtt{B}\}$

$$\begin{tikzcd} \mathtt{B} & 19 \\ \mathtt{A} \arrow[u] & 7 \arrow[u] \end{tikzcd}$$

$\mathtt{A} \sqcup 7 = \ldots$

$\{\mathtt{A}, \top, 9, \mathtt{B}, \bot\}$

$$\begin{tikzcd} & \top & \\ \mathtt{B} \arrow[ur] & & 9 \arrow[ul] \\ \mathtt{A} \arrow[u] & & \\ & \bot \arrow[ul] \arrow[uur] & \end{tikzcd}$$

$\mathtt{A} \sqcup 9 = \ldots$
$\mathtt{B} \sqcap 9 = \ldots$

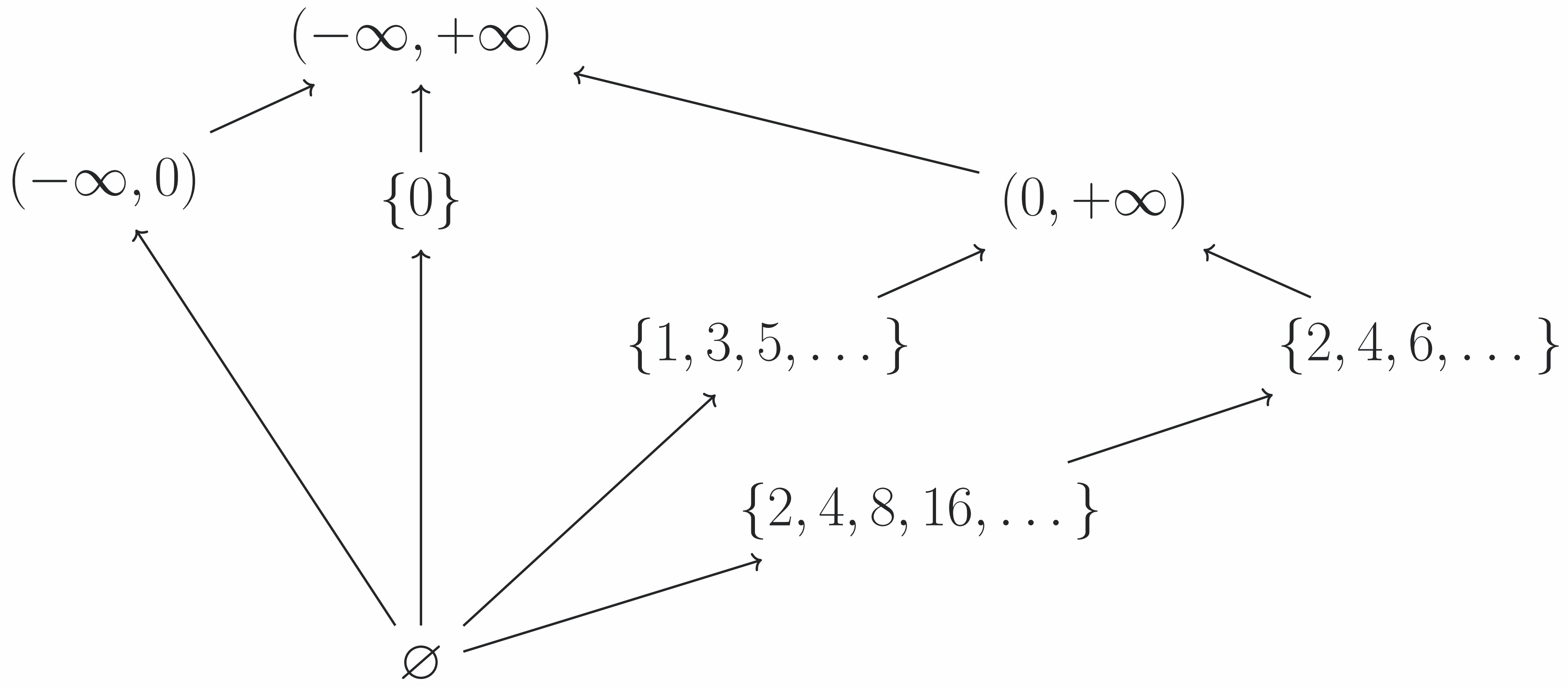
$\{\top, \bot, a, b, c, d\}$

$$\begin{tikzcd} & \top & \\ a \arrow[ur] & & c \arrow[ul] \\ b \arrow[u] \arrow[urr] & & d \arrow[u] \arrow[ull] \\ & \bot \arrow[ul] \arrow[ur] & \end{tikzcd}$$

$b \sqcup d = \ldots$
$a \sqcap c = \ldots$

# Intervals

A lattice may be used to represent intervals in a set of values, e.g. in $\mathbb{Z}$:




Partial order is $\in$.

Chapter #4:

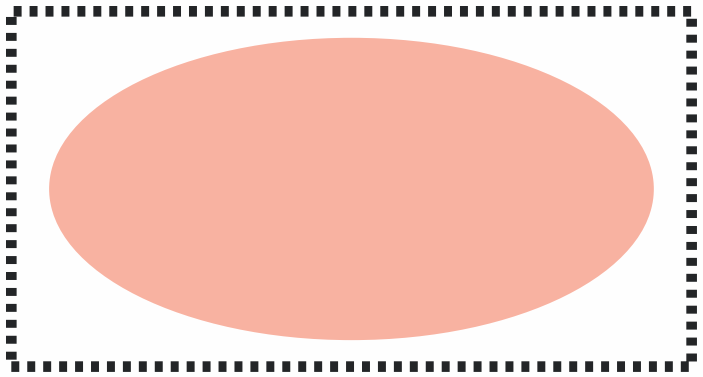
# Abstract Interpretation

# What is it about?

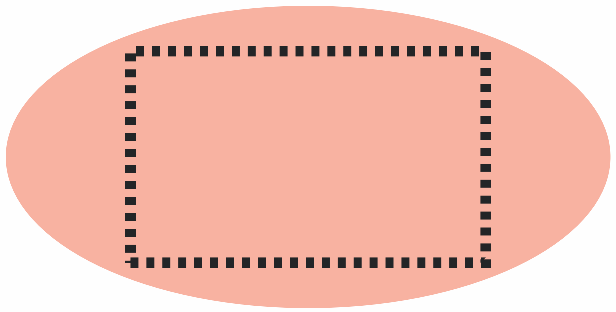
There is a compromise to be made between the precision of the analysis and its decidability (computability), or tractability (computational cost).

# Over and Under Approximation

What is the square of this oval?



1) **over**-approximation



2) **under**-approximation

# Abstraction and Concretization

Concrete domain $C$

$(\mathbb{Z}, <)$

$$\begin{tikzcd}
12 \\
7 \arrow[u] \\
4 \arrow[u] \\
0 \arrow[u] \\
-9 \arrow[u]
\end{tikzcd}$$

Abstract domain $A$

$(\{s|\{n|n \in \mathbb{N}\}\}, \in)$

$$\begin{tikzcd}
& \mathbb{N} & \\
2x \arrow[ur] & & 3x \arrow[ul] \\
2^x \arrow[u] & & 6x \arrow[u] \arrow[ull] \\
& \varnothing \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Abstraction function:

$\alpha(c) \to a$

Concretization function:

$\gamma(a) \to c$

Domains must be related through Galois connection:

$\forall c \in C, \forall a \in A$

$\alpha(c) \sqsubseteq a \iff c \sqsubseteq \gamma(a)$

Are they?

# Abstract Semantics (Transformers)

**Abstract domain:**

$$
\begin{tikzcd}
& \mathbb{N} & \\
2x \arrow[ur] & & 3x \arrow[ul] \\
2^x \arrow[u] & & 6x \arrow[u] \arrow[ull] \\
& \varnothing \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

**Transformers:**

$\mathbb{N} + \mathbb{N} = \dots$

$2x + 2x = \dots$

$2x + 3x = \dots$

$2x \times 3x = \dots$

$\varnothing + 2x = \dots$

**Concrete counterparts:**

$1024 + 1 = \dots$

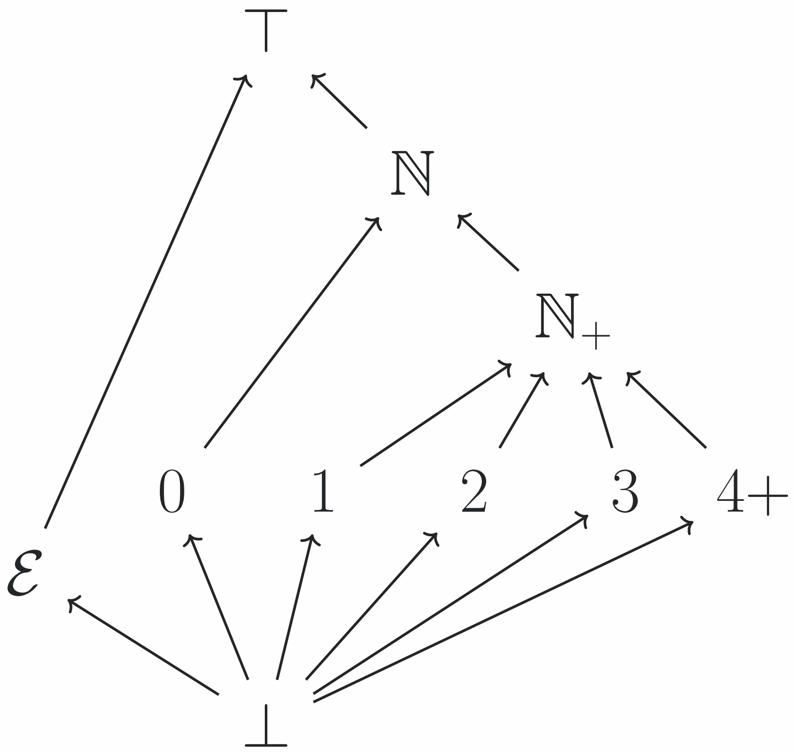
$46 + 4 = \dots$

$8 + 9 = \dots$

$6 \times 12 = \dots$

$-1 + 4 = \dots$

# Widening and Narrowing




$$0 \bigtriangledown 1 = \dots$$

$$1 \bigtriangledown \mathbb{N}_+ = \dots$$

$$0 \bigtriangledown \mathbb{N}_+ = \dots$$

$$1 \bigtriangleup \mathbb{N}_+ = \dots$$

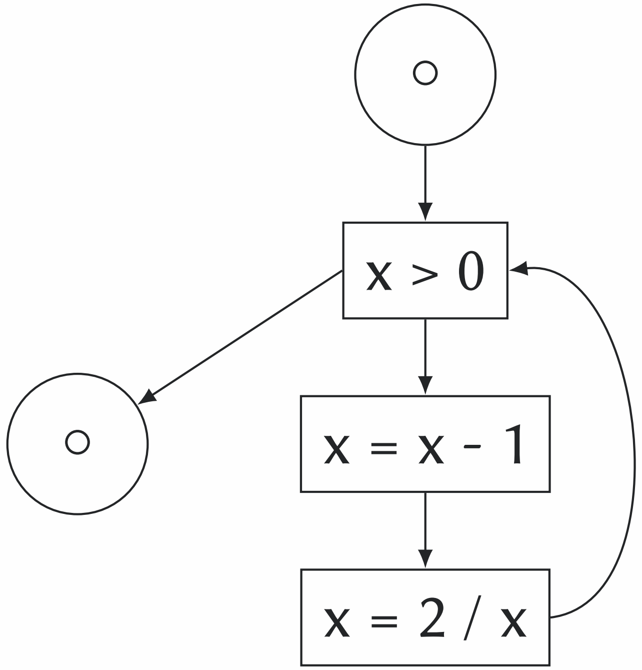
$$0 \bigtriangleup 1 = \dots$$

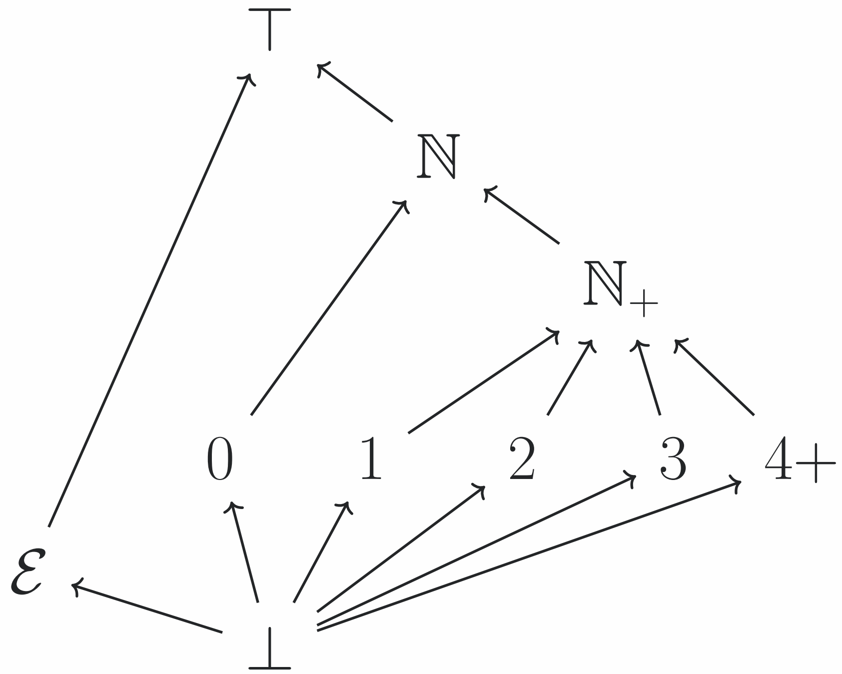
$$3 \bigtriangleup 4+ = \dots$$

# Fixed-Point Computation

Fixed-Point Computation is a repeated symbolic execution of the program using abstract semantics until approximation reaches an equilibrium.

```
int f(int x) {
  while x > 0 {
    x = x - 1;
    x = 2 / x;
  }
  return x;
}
```



Chapter #5:

# Further Reading/Watching

Lecture by Patrick Cousot, on YouTube

Mozilla wiki page on Abstract Interpretation.

Slide deck of Işil Dillig.

Introduction to Abstract Interpretation by Bruno Blanchet.