



Static Analysis using Abstract Interpretation

Maxime Arthaud

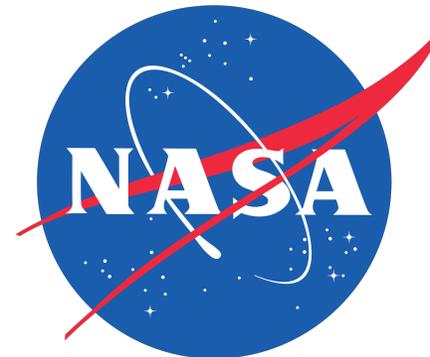
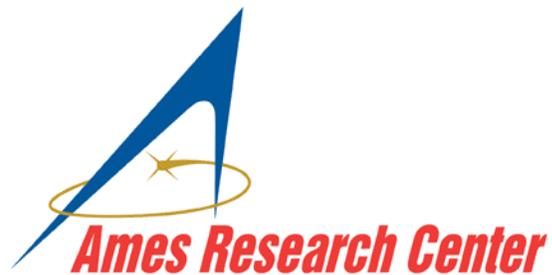


Table of contents

1. Introduction
2. IKOS
3. Abstract Interpretation

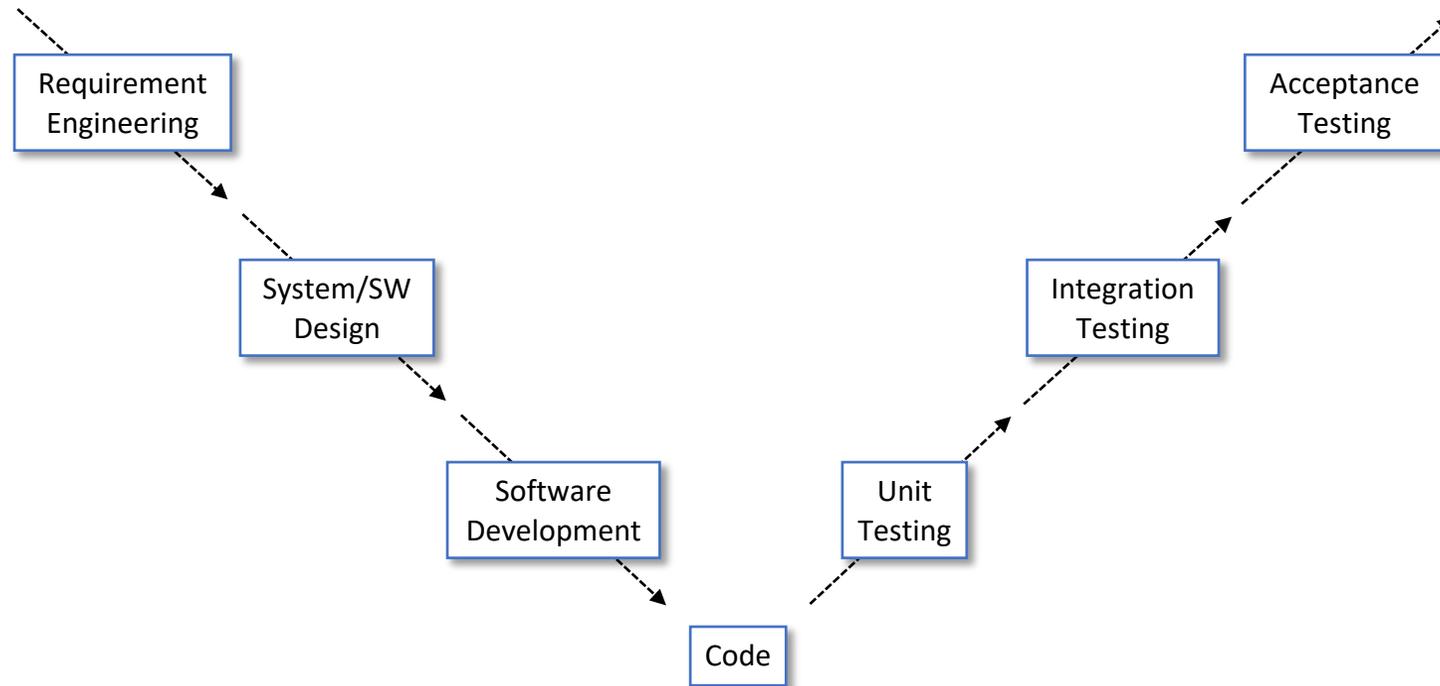
Table of contents

1. Introduction
 - A. Software development
 - B. Validation and Verification
 - C. Formal Verification
 - D. Static Analysis
2. IKOS
3. Abstract Interpretation

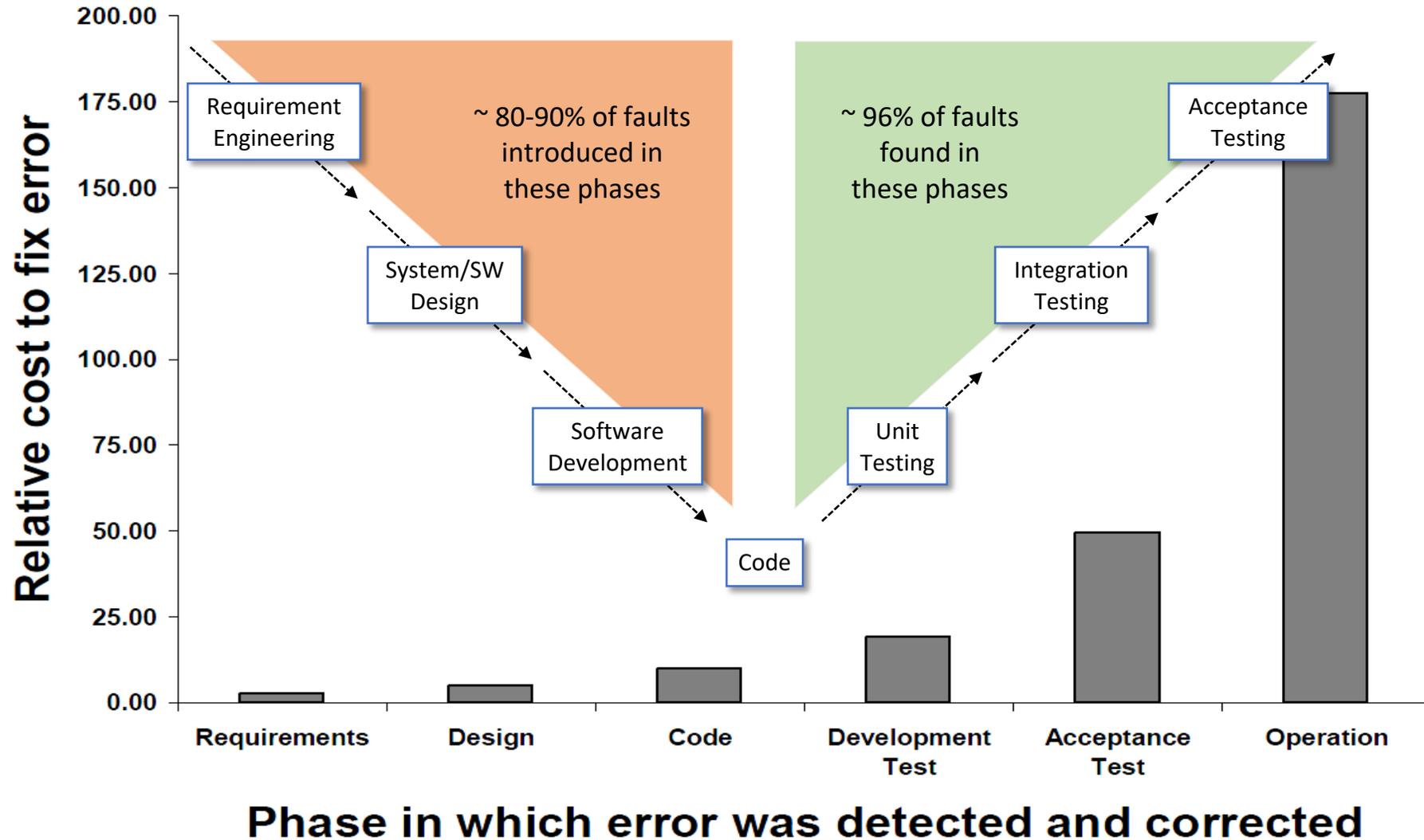
Software development

- Software represent **more than half** of the development **cost** of an aircraft
- **Safety Critical** : Failure is not an option
- Regulated by international standards : **DO-178 rev. B/C**
- Robust Software Engineering

V Model



Cost Analysis



Validation and Verification

- **Validation:**

The software **meets** the **needs** of the user

→ Are we building the right thing?

- **Verification:**

The software **meets** the **specification**

→ Are we building it right?

Formal Verification

- **Formal Verification:**

Prove the correctness of a program with respect to a **formal specification**

- **Formal Specification:**

Mathematical description of a software

Properties

- **Safety property:**
Something **bad** will **never** happen
- **Liveness property:**
Something **good** will **eventually** happen

Formal Methods

- **Abstract Interpretation:**
 - Compute an over approximation of all the reachable states
- **Model Checking:**
 - Explore all the reachable states
- **Symbolic Execution:**
 - Explore interesting paths with symbolic inputs
- **Theorem Proving:**
 - Prove properties manually or with heuristic algorithms
- Etc...

Static Analysis

- **Static Analysis:**

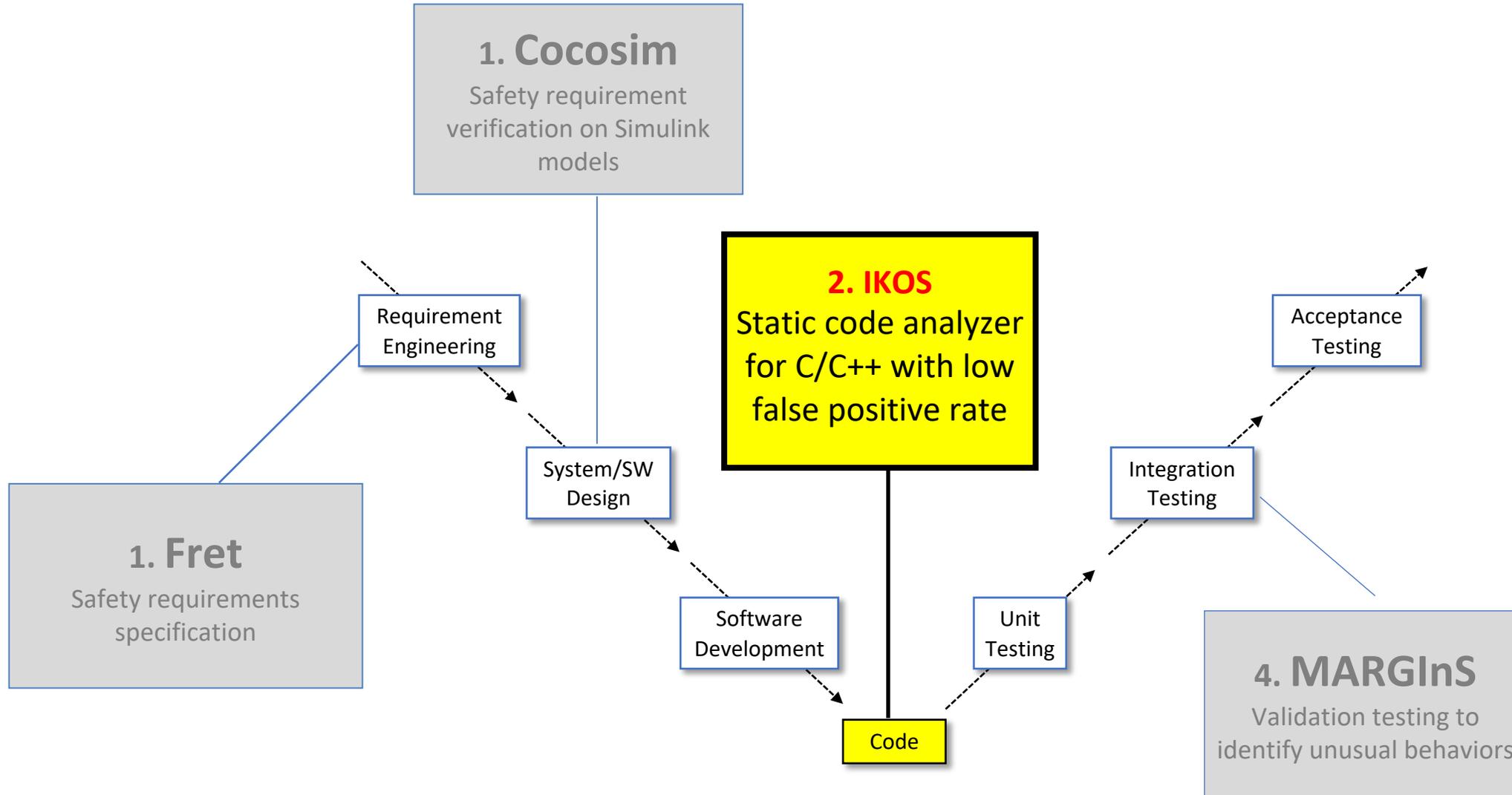
Analysis of a software **without** actually **executing** it

- Opposite of Dynamic Analysis
- Can be performed on the **source code** or **binary code**
- Objectives:
 - Find **runtime errors**
 - Measure **metrics**
 - Reverse engineering
 - Etc...

Table of contents

1. Introduction
2. **IKOS**
3. Abstract Interpretation

IKOS



IKOS

IKOS performs a **compile-time** analysis of a C/C++ source code.
It can **detect** or **prove the absence** of **runtime errors**.

```
int tab[10];  
  
for (int i = 0; i < 10; i++) {  
    tab[i] = i * i;  
}
```

C/C++ code

IKOS
Static
Analyzer

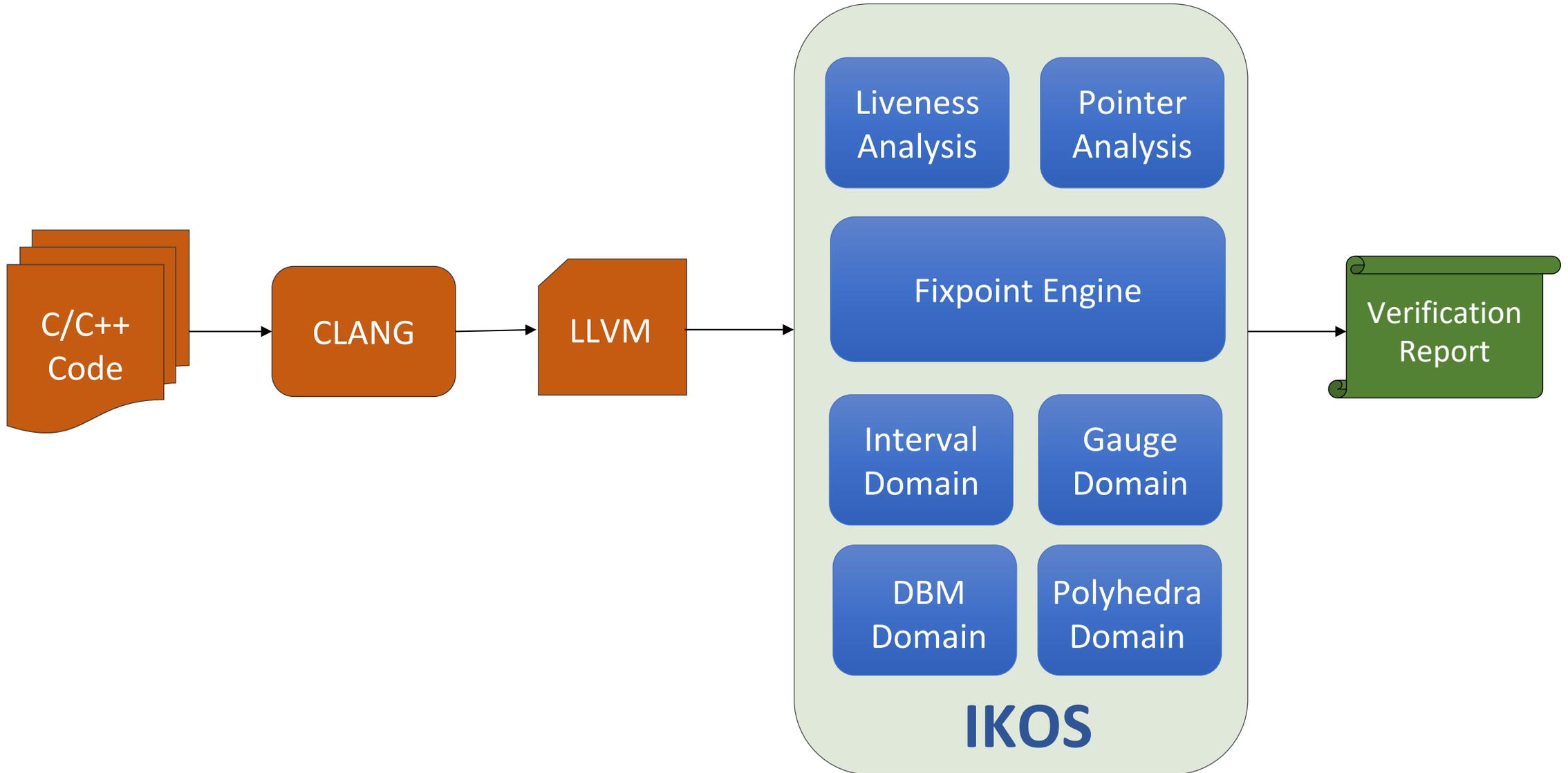
List of (possible) runtime errors:

- Buffer Overflows
- Null pointers
- Division by Zero
- Uninitialized Variables
- Assertion Prover
- Etc.

IKOS is **NOT** a code style checker

IKOS is **NOT** a compiler: It can detect errors that compilers cannot catch

IKOS Design



Verification Report

- **Safe**: The instruction is **proven free of runtime errors**
- **Error**: The instruction **always produces a runtime error**
- **Warning**:
 - The instruction **can produce an error** depending on the input
 - The instruction is **safe** but IKOS could **not prove it** (also called **false positive**)

Example

```
int tab[10];  
  
for (int i = 0; i < 10; i++) {  
    tab[i] = i * i;  
}
```

- The analysis discovers program properties: $0 \leq i \leq 9$

Example

```
int tab[10];  
  
for (int i = 0; i < 10; i++) {  
    tab[i] = i * i;  
}
```

Access within bounds?



- The analysis discovers program properties: $0 \leq i \leq 9$
- The verification uses the properties discovered:
 - Array-bound compliance
 - Check that array **tab** has at least 10 elements

IKOS Checks

- Buffer overflow
- Division by zero
- Null pointer dereference
- Assertion prover
- Unaligned pointer
- Uninitialized variable
- Integer overflow (signed, unsigned)
- Invalid bit shift
- Invalid pointer comparison
- Invalid function pointer call
- Dead code
- Double free and Invalid lifetime

IKOS Abstract Domains

Domain	Constraints	Complexity
Interval	$x \in [a, b]$	n
Congruence	$x \in aZ+b$	n
Gauge	$x \in [a*i + b*k + \dots, a'*i + b'*k + \dots]$	$K*n$
Difference Bound Matrices	$x - y \in [a, b]$	n^3
APRON Octagon	$x \pm y \in [a, b]$	n^3
APRON Polka Polyhedra	$a*x + b*y + \dots + c \leq 0$	Exponential
APRON PPL Polyhedra	$a*x + b*y + \dots + c \leq 0$	Exponential
Variable Packing of	n

Live demo

IKOS Installation

- Supported platforms:
 - Mac OS
 - Linux
 - Windows (using *MinGW*)
- **Dependencies** can be installed with a **package manager** ([brew](#), [apt-get](#), [yum](#), ..)
- **Installation instructions** for each platform available in: [doc/install/](#)
- **Bootstrap script** for **non-admin** installations: downloads and compiles all missing dependencies (*slow*)

IKOS Usage

- Analyze a single file: `ikos file.c`
 - Runs the analysis
 - Prints the results
 - Generates an output database containing the analysis results: `output.db`
- Analyze a whole project:
 - `ikos-scan make`
 - `ikos program.bc`
- Generate a report from an output database: `ikos-report output.db`
- Examine the results in a graphical interface: `ikos-view output.db`

IKOS-SCAN

- Analyze a whole project with: `ikos-scan <command>`
- It compiles all executables to LLVM bitcode: `program.bc`
- It runs IKOS on the LLVM bitcode: `ikos program.bc`
- Works with most build systems: `Make`, `CMake`, `Autoconf`, etc...
- Works by overriding environment variables: `CC`, `CXX`, `LD`

IKOS-SCAN

Live demo

Analyzing a library

- The analysis needs an **entry point** (i.e, `main`)
- **Workaround:** create a small program that uses the library
- Analyze a program with a specific entry point: `ikos file.bc -e=MyMain`

IKOS-VIEW

- **Graphical interface** to examine the analysis results
- Starts a **web server** in the terminal, opens the default **browser**
- `ikos-view output.db`

IKOS-VIEW

Live demo

IKOS Abstract Domains Guidelines

- Start with **fast** but **imprecise** domain
- Go towards **slow** but **precise** domain
- Stop when the analysis is too slow for your use case
- Recommended order:
 - Interval: `-d=interval`
 - Gauge + Interval + Congruence: `-d=gauge-interval-congruence`
 - Variable Packing DBM: `-d=var-pack-dbm`
 - Variable Packing Polyhedra: `-d=var-pack-apron-ppl-polyhedra`

IKOS Assumptions

- The source code is compiled with Clang for the **host architecture**
- Clang defines:
 - The data model (size of types)
 - The memory layout (alignments)
 - The endianness
 - The semantic of floating points
 - Etc...

IKOS Assumptions

- The program is **single-threaded**
- The program does **not** receive **signals** or **interrupts**
- **Unknown extern functions:**
 - Do not update global variables
 - Can write on their pointer arguments
 - Do not call user-defined functions (no callbacks)
- **Assembly code** is treated as a call to an unknown extern function
- **Recursive functions** can update any value in memory

False positives

- **False positive:** invalid warning
- Objective: **low rate of false positives**
- Common source of false positives:
 - Unknown library functions
 - “Bad” code patterns
 - Imprecision of the analysis

Modeling library functions

- The analyzer does **not** require the **libraries** used by your program
- Unknown library functions will trigger a warning ("ignored call side effect" in [ikos-view](#))
- Modeling library functions can **reduce the number of warnings**
- Write "stubs": fake implementations of library functions

Modeling library functions

```
#include <ikos/analyzer/intrinsic.h>

char* fgets(char* restrict str,
            int size,
            FILE* restrict stream) {
    __ikos_assert(size >= 0);
    __ikos_forget_mem(stream, sizeof(FILE));
    __ikos_abstract_mem(str, size);
    errno = __ikos_nondet_int();
    return __ikos_nondet_int() ? str : NULL;
}
```

IKOS Annotations

- Annotating your source code can **reduce the number of warnings**
- List of intrinsic functions: [analyzer/include/ikos/analyzer/intrinsic.h](#)
 - `__ikos_assert(condition)`
 - `__ikos_assume(condition)`
 - `__ikos_nondet_int()`
 - `__ikos_check_mem_access(ptr, size)`
 - `__ikos_assume_mem_size(ptr, size)`
 - `__ikos_forget_mem(ptr, size)`
 - `__ikos_abstract_mem(ptr, size)`
 - `__ikos_print_values(description, var)`

IKOS Annotations

```
ret = talg->parse_algoid_params(buf, param_len, param);
```

IKOS Annotations

```
ret = talg->parse_algoid_params(buf, param_len, param);
```

```
int (*fun)(const u8*, u16, alg_param*) =  
    talg->parse_algoid_params;  
__ikos_assume(fun == parse_algoid_params_generic ||  
              fun == parse_algoid_params_ecdsa_with ||  
              fun == parse_algoid_params_ecPublicKey ||  
              fun == parse_algoid_params_rsa);  
ret = fun(buf, param_len, param);
```

Bad code pattern (1)

```
CommandResult = XXX();
if (CommandResult == TRUE) {
    FilenameState = YYY();
    if (FilenameState == FM_NAME_IS_INVALID) {
        CommandResult = FALSE;
    }
}
if (CommandResult == TRUE) {
    CommandResult = ZZZ();
}
if (CommandResult == TRUE) {
    // ...
}
return CommandResult;
```

Bad code pattern (1)

- Bad readability
- Prone to errors
- Hard for static analyzers
- Please use “early return on errors”

Bad code pattern (1)

```
CommandResult = XXX();  
if (!CommandResult) {  
    return FALSE;  
}  
CommandResult = YYY();  
if (CommandResult == FM_NAME_IS_INVALID) {  
    return FALSE;  
}  
CommandResult = ZZZ();  
if (!CommandResult) {  
    return FALSE;  
}  
// ...  
return TRUE;
```

Bad code pattern (2)

- Single global variable containing everything

```
AppData_t g;

typedef struct {
    PipeId_t CmdPipeId;
    uint16 usCmdPipeDepth;
    char cCmdPipeName[OS_MAX_API_NAME];
    int32 ulfd;
    uint32 uiRunStatus;
    // ...
    uint8 lastCmdBchErrorStatus;
} AppData_t;
```

Bad code pattern (2)

- Makes the buffer overflow analysis harder
- Please split it into different global variables

Bad code pattern (3)

- Small integers for loop counters

```
void f(uint16_t n) {  
    for (uint16_t i = 0; i < n; i++) {  
        // ...  
    }  
}
```

Bad code pattern (3)

- Small integers for loop counters

```
void f(uint16_t n) {  
    for (uint16_t i = 0; i < n; i++) {  
        // ...  
    }  
}
```

- Integer promotion rules of C

```
void f(uint16_t n) {  
    for (uint16_t i = 0;  
        (unsigned int)i < (unsigned int)n;  
        i = (uint16_t)((unsigned int)i + 1)) {  
        // ...  
    }  
}
```

Bad code pattern (3)

- Creates temporary variables in the LLVM bitcode
- Leads to imprecision of the analysis
- Please use `size_t` (or `int`) for loop indexes

Imprecision

- Initialization functions returning an error code

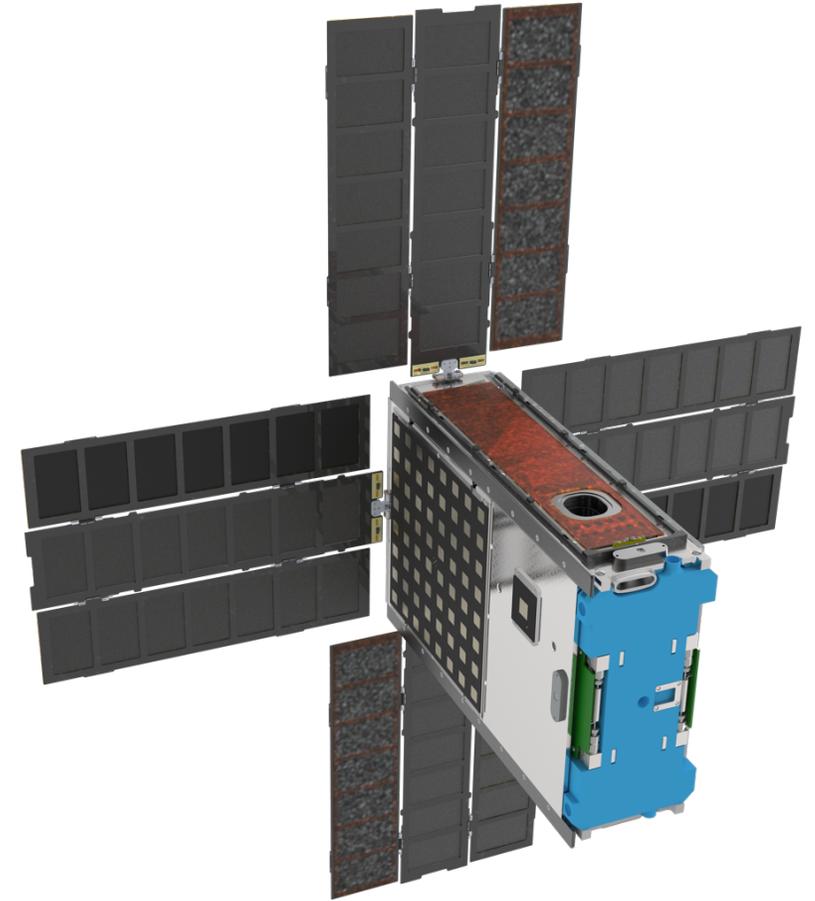
```
int Init(void) {  
  
    int status = Register();  
    if (status != SUCCESS) {  
        return status;  
    }  
  
    status = InitEvent();  
    if (status != SUCCESS) {  
        return status;  
    }  
  
    // ...  
}
```

Imprecision

- Imprecision due to the abstract union in the analysis
- Analysis option: `--partitioning=return`

Success Story: BioSentinel

- Space biology mission
- CubeSat spacecraft
- Developed at NASA Ames, in collaboration with JPL, JSC, MSFC
- Flight software built on top of CFS



Success Story: BioSentinel

- Each application was analyzed with IKOS
- The CFE framework was modeled to improve the analysis (~ 1200 LOC)
- Low rate of warnings: 1.31% in average
- Found ~ 17 real bugs

Success Story: BioSentinel

Application	Abstract Domain	Time	Errors	Warnings	Warnings%	Checks
adio	var-pack-dbm	1 min 6.92 sec	0	1	0.07%	1334
brdio	var-pack-dbm	8.02 sec	0	8	0.97%	818
ci	var-pack-dbm	19.98 sec	0	6	0.65%	923
comio	var-pack-dbm	1 min 4.83 sec	0	4	0.26%	1494
epsio	var-pack-dbm	30.64 sec	0	5	0.42%	1181
letio	var-pack-dbm	24.33 sec	0	18	1.64%	1095
ms	interval	0.16 sec	0	0	0%	444
saio	var-pack-dbm	22.35 sec	0	8	0.64%	1246
sensio	var-pack-dbm	4.67 sec	0	79	9.56%	826
spe	interval	0.16 sec	0	0	0%	445
thrio	var-pack-dbm	19.33 sec	0	4	0.38%	1043
to	var-pack-dbm	2 min 18.32 sec	0	33	1.98%	1666
xactio	var-pack-dbm	22.18 sec	0	6	0.51%	1165

BioSentinel Bug (1)

warning: Possible buffer overflow, pointer '&FrameBuffer[0]' with offset 0 bytes points to global variable 'FrameBuffer' of size 4096 bytes

```
ssize_t numbytes = read(fd, &FrameBufferRaw[0], 4096);

if (numbytes < 0) {
    return ERROR;
}

// ...

numbytes -= 9; // Integer overflow

memcpy(/*dst*/ FrameBuffer, /*src*/ FrameBufferRaw, /*size*/ numbytes);
^~~~~~
```

BioSentinel Bug (2)

warning: Possible buffer overflow, pointer '&cmd[n + 2]' accesses 1 bytes at offset between 8 and 16 bytes of local variable 'cmd' of size 16 bytes

```
uint8_t cmd[16];
uint8_t n;
// ...
switch(cmd_request) {
    case CMD_OPEN:
        n = CMD_OUT + CMD_OPEN; // 6 + 8 = 14
        break;
    // ...
}
// ...
cmd[n + 2] = 0; // 14 + 2 = 16
```

Guidelines

- Use a lightweight static analyzer first: `cppcheck`, `clang-tidy`, `pvs-studio`, etc.
- Use `ikos-scan` to generate the llvm bitcode (`.bc`): `ikos-scan make`
- Use `ikos` on the llvm bitcode (`.bc`): `ikos program.bc`
- Try different abstract domains: `ikos -d=var-pack-dbm program.bc`
- Use `ikos-view` to examine the results: `ikos-view output.db`
- (Optional) Model key library functions
- (Optional) Annotate the code
- (Optional) Avoid “bad” patterns
- (Optional) Add `ikos` in your continuous build system

IKOS at a glance

- IKOS is a **static analyzer** for **C/C++** targeting **safety critical** software
- IKOS is **open source**: <https://github.com/NASA-SW-VnV/ikos>
- Contact: ikos@lists.nasa.gov

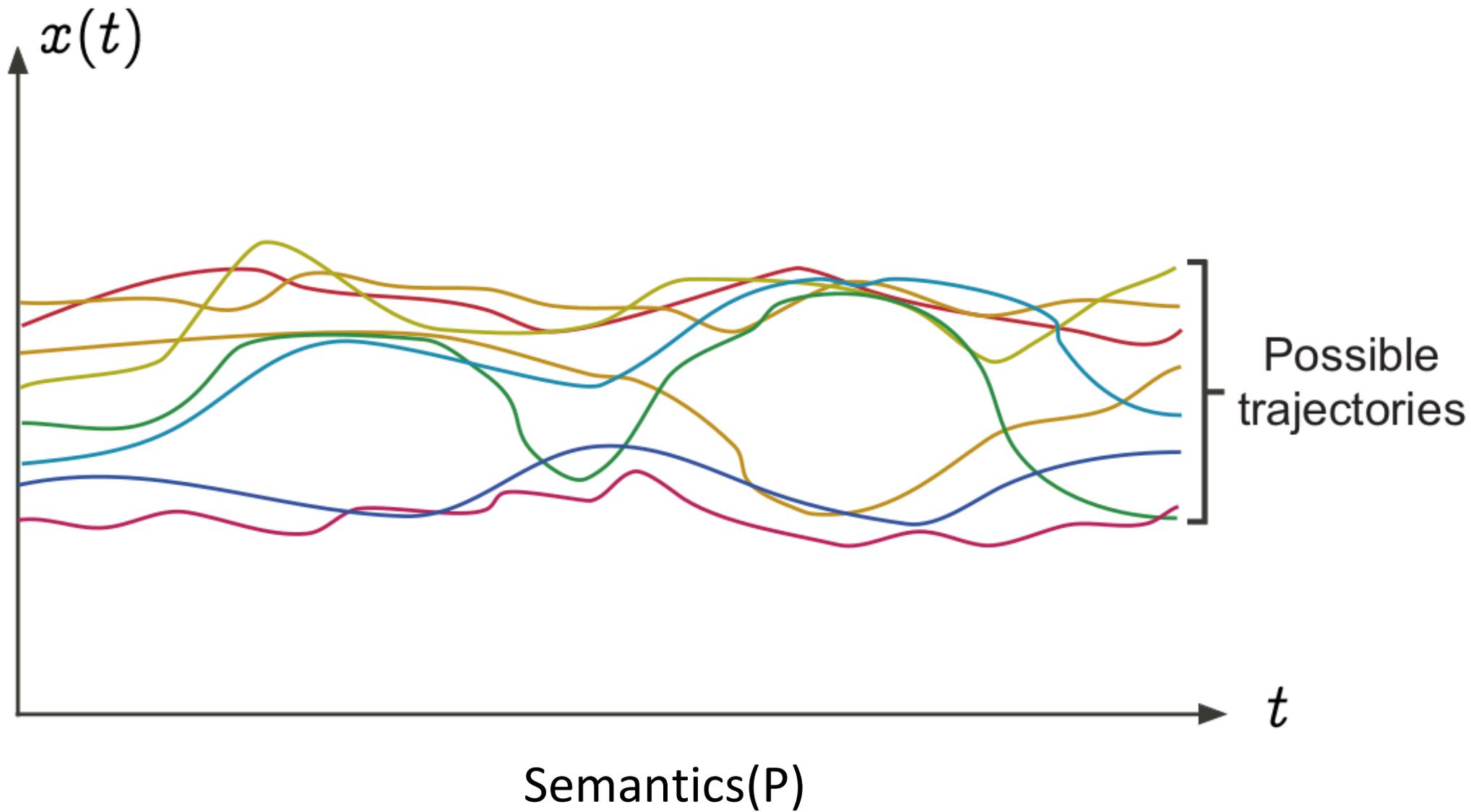
Table of contents

1. Introduction
2. IKOS
3. Abstract Interpretation
 - A. Overview
 - B. Concrete semantic
 - C. Abstract semantic
 - D. Interval domain
 - E. Convergence Acceleration

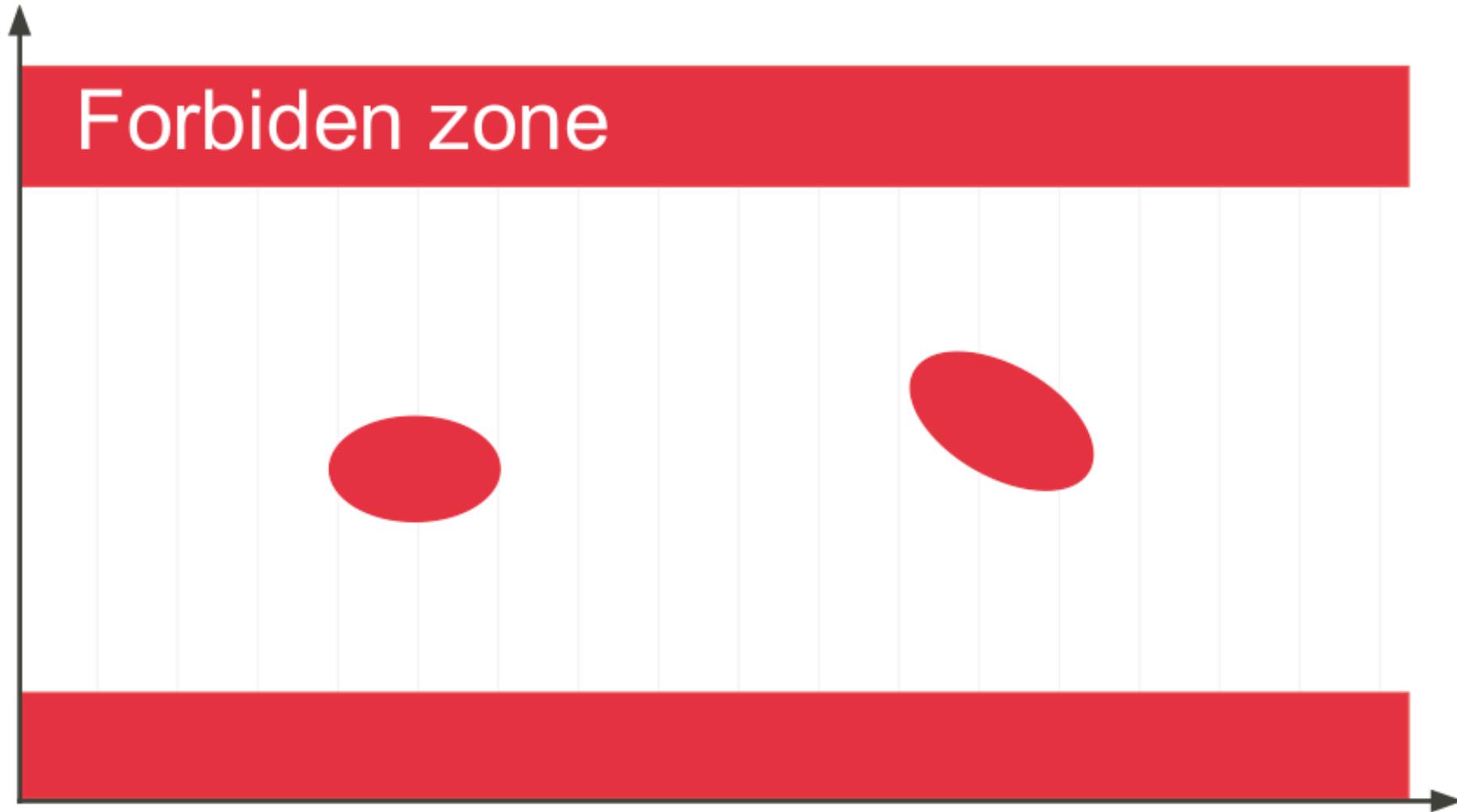
Abstract Interpretation

- Mathematical framework
- **Approximation** of the reachable **states** of a program
- **Fully automated**: No user interaction
- **Sound**: Cannot miss a bug
- Formalized by Patrick and Radhia Cousot in the late 1970s

Overview



Overview



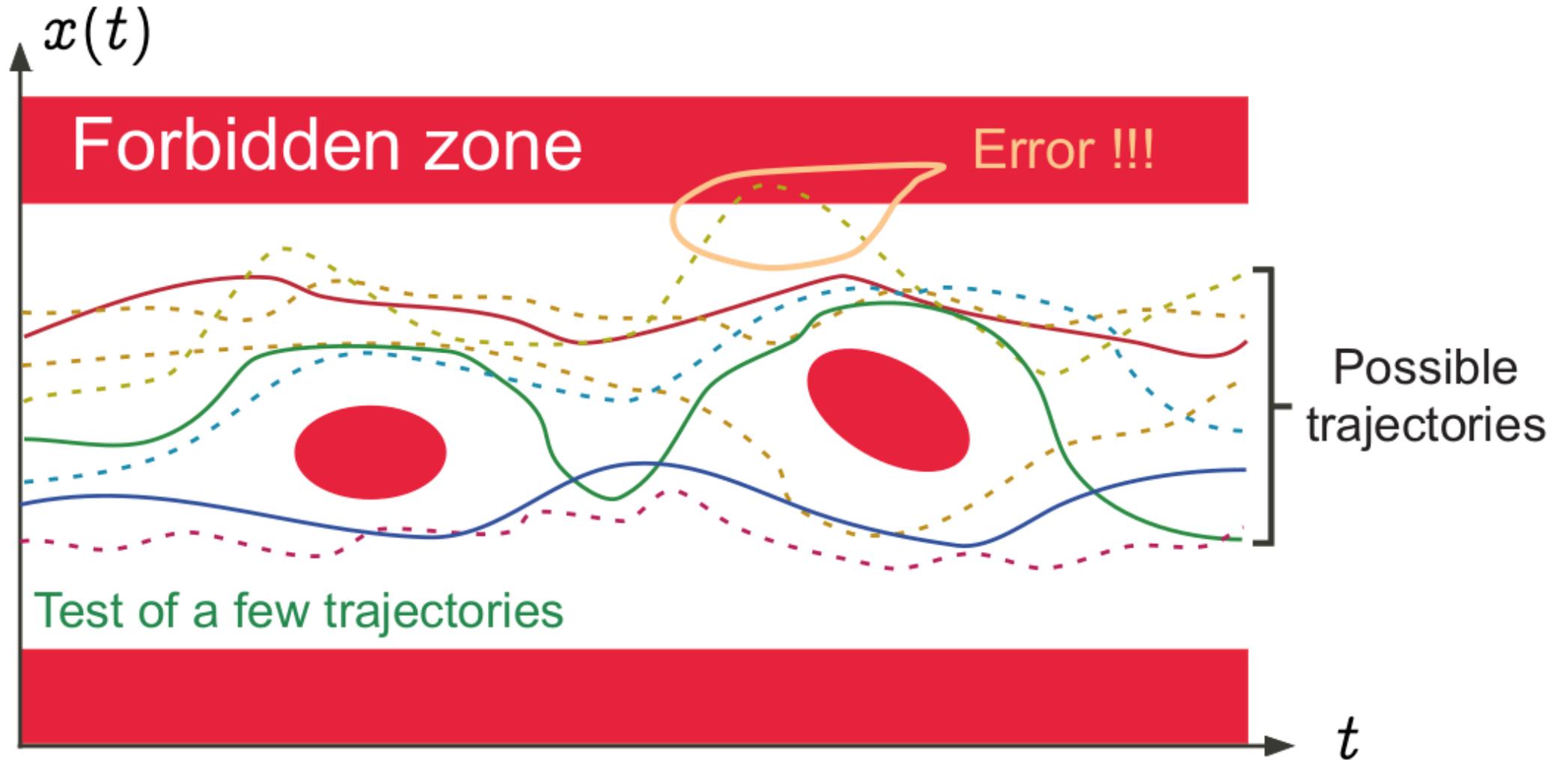
Specification(P)

Overview



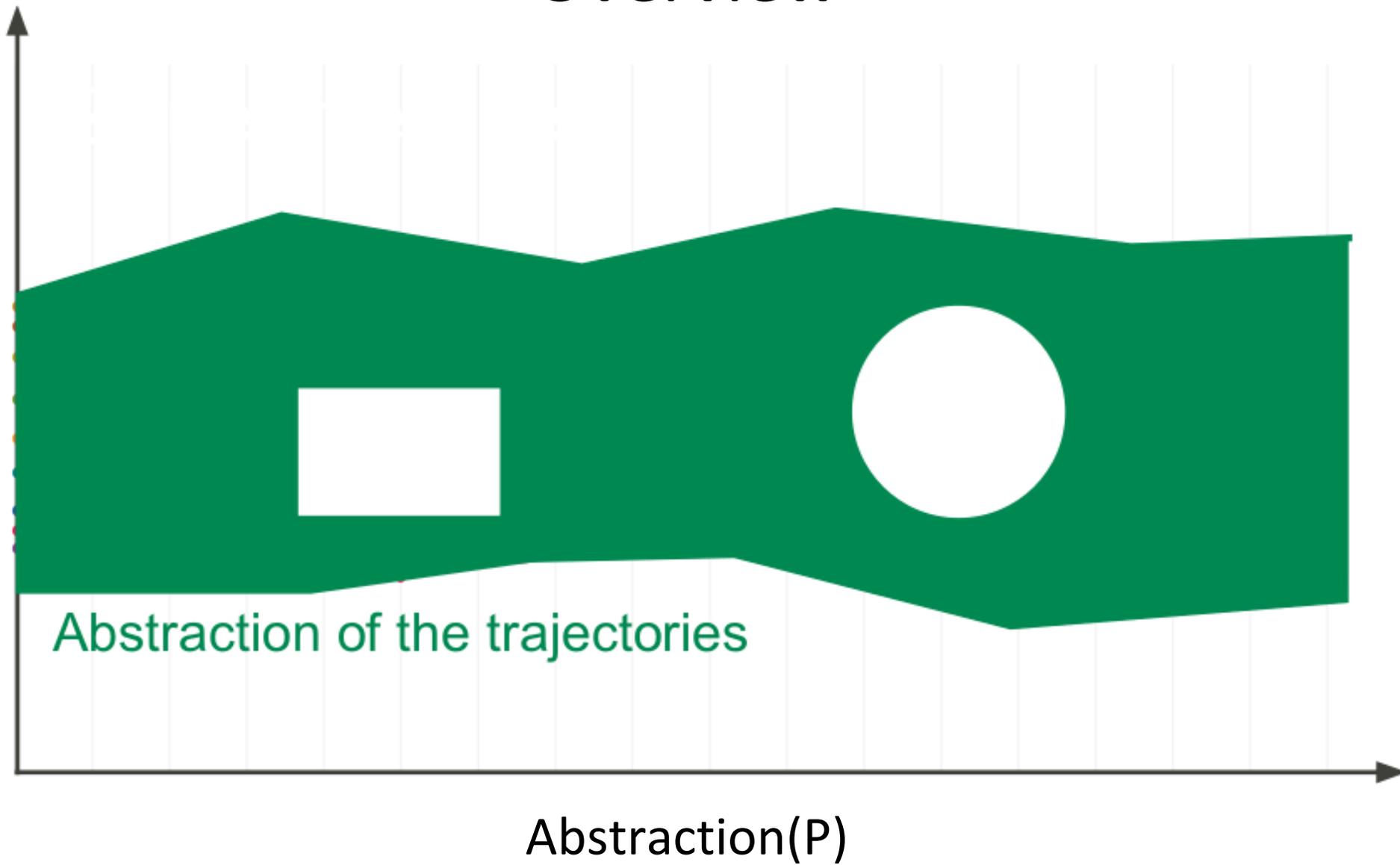
Semantics(P) \subseteq Specification(P) ?

Overview



Using Testing 

Overview

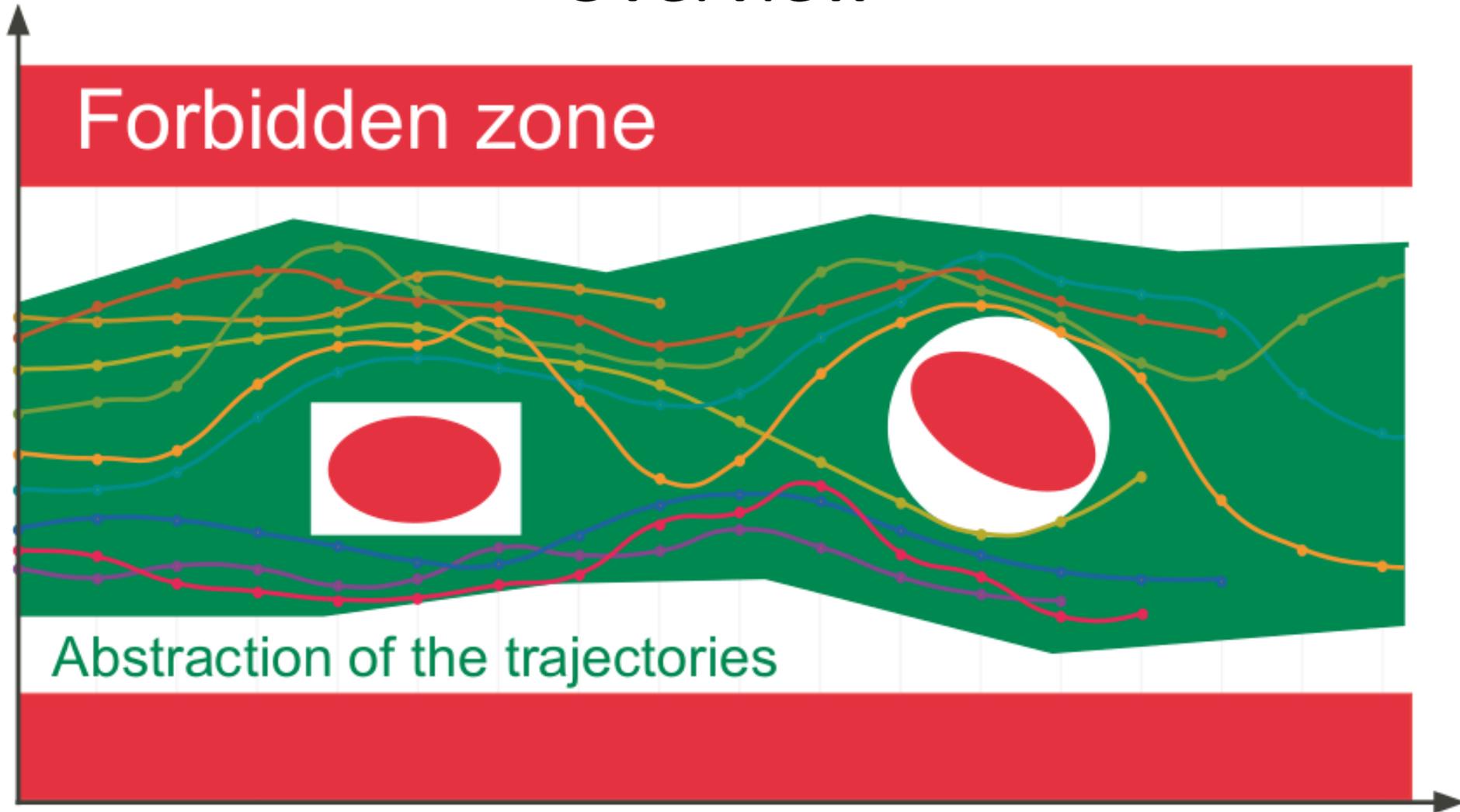


Overview



$$\text{Abstraction}(P) \subseteq \text{Specification}(P)$$

Overview



$\text{Semantics}(P) \subseteq \text{Abstraction}(P) \subseteq \text{Specification}(P)$ ✓

Toy language - Syntax

```
stmt ::= | v = expr;  
      | stmt stmt  
      | if (expr > 0) { stmt }  
      | else { stmt }  
      | while (expr > 0) { stmt }  
  
expr ::= | v ∈ V  
      | n ∈ ℤ  
      | rand(a, b)  
      | expr + expr  
      | expr - expr  
      | expr * expr  
      | expr / expr
```

V set of variables

\mathbb{Z} set of integers

rand(a, b) represents an integer between **a** and **b** (simulate an input)

Toy language - Example

```
x = rand(0, 12);  
y = 42;  
while (x > 0) {  
    x = x - 2;  
    y = y + 4;  
}
```

An execution:

(values at the beginning of the loop)

x		7	5	3	1	-1
y		42	46	50	54	58

Toy language - Example

```
x = rand(0, 12);  
y = 42;  
while (x > 0) {  
    x = x - 2;  
    y = y + 4;  
}
```

An execution:

(values at the beginning of the loop)

x		7	5	3	1	-1
y		42	46	50	54	58

Notes:

- A very **simple** language, no functions, no arrays, ...
- But it is an **imperative language** like C
- It is actually a **subset of C**
- It can compute everything – **Turing complete**

Toy language - Semantic

- We need to define the semantic of our language
- Also called **Formal Specification**
- **Collective** semantic:
Mathematical definition of the reachable states of a given program

Control flow graph

A **control flow graph** is a triplet (L, O, A) with a set of program points L , an entry point $O \in L$ and a set of edges $A \subseteq L \times \text{command} \times L$ with:

command ::= | $v = \text{expr}$
 | $\text{expr} > 0$

Control flow graph

A **control flow graph** is a triplet (L, O, A) with a set of program points L , an entry point $O \in L$ and a set of edges $A \subseteq L \times \text{command} \times L$ with:

command ::= | $v = \text{expr}$
 | $\text{expr} > 0$

```
0x = rand(0, 12); 1y = 42;  
while 2(x > 0) {  
    3x = x - 2;  
    4y = y + 4;  
}5
```

Control flow graph

A **control flow graph** is a triplet (L, O, A) with a set of program points L , an entry point $O \in L$ and a set of edges $A \subseteq L \times \text{command} \times L$ with:

command ::= | $v = \text{expr}$
 | $\text{expr} > 0$

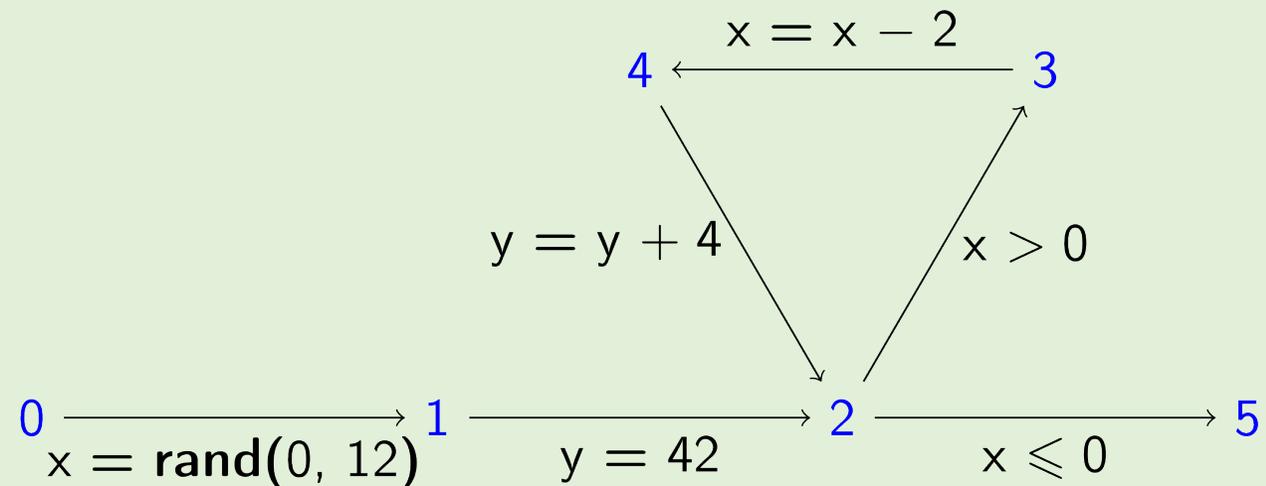
```
0 x = rand(0, 12); 1 y = 42;
```

```
while 2 (x > 0) {
```

```
    3 x = x - 2;
```

```
    4 y = y + 4;
```

```
} 5
```



Semantic of Expressions

Semantic of expressions: $\llbracket e \rrbracket_E : (\mathbb{V} \rightarrow \mathbb{Z}) \rightarrow \mathcal{P}(\mathbb{Z})$

$$\llbracket v \rrbracket_E (\rho) = \{\rho(v)\}$$

$$\llbracket n \rrbracket_E (\rho) = \{n\}$$

$$\llbracket \mathbf{rand}(n_1, n_2) \rrbracket_E (\rho) = \{n \in \mathbb{Z} \mid n_1 \leq n \leq n_2\}$$

$$\llbracket e_1 + e_2 \rrbracket_E (\rho) = \{n_1 + n_2 \mid n_1 \in \llbracket e_1 \rrbracket_E (\rho) \wedge n_2 \in \llbracket e_2 \rrbracket_E (\rho)\}$$

...

Semantic of Expressions

Semantic of expressions: $\llbracket e \rrbracket_E : (\mathbb{V} \rightarrow \mathbb{Z}) \rightarrow \mathcal{P}(\mathbb{Z})$

$$\llbracket v \rrbracket_E (\rho) = \{\rho(v)\}$$

$$\llbracket n \rrbracket_E (\rho) = \{n\}$$

$$\llbracket \mathbf{rand}(n_1, n_2) \rrbracket_E (\rho) = \{n \in \mathbb{Z} \mid n_1 \leq n \leq n_2\}$$

$$\llbracket e_1 + e_2 \rrbracket_E (\rho) = \{n_1 + n_2 \mid n_1 \in \llbracket e_1 \rrbracket_E (\rho) \wedge n_2 \in \llbracket e_2 \rrbracket_E (\rho)\}$$

...

An environment ρ is a function $\mathbb{V} \rightarrow \mathbb{Z}$ that associates a value to each variable

Errors

Notes on errors:

We can reach 2 kind of errors during the execution:

Errors

Notes on errors:

We can reach 2 kind of errors during the execution:

- `rand(n1, n2)` with $n_1 > n_2$:

$$\llbracket \text{rand}(n_1, n_2) \rrbracket_E = \{x \in \mathbb{Z} \mid n_1 \leq x \leq n_2\} = \emptyset;$$

Errors

Notes on errors:

We can reach 2 kind of errors during the execution:

- `rand(n1, n2)` with $n_1 > n_2$:

$$\llbracket \text{rand}(n_1, n_2) \rrbracket_{\text{E}} = \{x \in \mathbb{Z} \mid n_1 \leq x \leq n_2\} = \emptyset ;$$

- Division by zero :

$$\llbracket e/0 \rrbracket_{\text{E}} = \emptyset .$$

Errors

Notes on errors:

We can reach 2 kind of errors during the execution:

- $\text{rand}(n_1, n_2)$ with $n_1 > n_2$:

$$\llbracket \text{rand}(n_1, n_2) \rrbracket_{\text{E}} = \{x \in \mathbb{Z} \mid n_1 \leq x \leq n_2\} = \emptyset ;$$

- Division by zero :

$$\llbracket e/0 \rrbracket_{\text{E}} = \emptyset.$$

We assume the program aborts on errors.

Semantic of commands

Semantic of commands: $\llbracket c \rrbracket_C : \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z}) \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$

$$\llbracket v = e \rrbracket_C (R) = \{\rho[v \mapsto n] \mid \rho \in R, n \in \llbracket e \rrbracket_E (\rho)\}$$

$$\llbracket e > 0 \rrbracket_C (R) = \{\rho \mid \rho \in R, \exists n \in \llbracket e \rrbracket_E (\rho), n > 0\}$$

Semantic of commands

Semantic of commands: $\llbracket c \rrbracket_C : \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z}) \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$

$$\llbracket v = e \rrbracket_C (R) = \{\rho[v \mapsto n] \mid \rho \in R, n \in \llbracket e \rrbracket_E (\rho)\}$$

$$\llbracket e > 0 \rrbracket_C (R) = \{\rho \mid \rho \in R, \exists n \in \llbracket e \rrbracket_E (\rho), n > 0\}$$

Note that $e \leq 0$ can be rewritten as $1 - e > 0$ (syntactic sugar)

Semantic of programs

Semantic of programs: $\llbracket (L, A) \rrbracket : L \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$

For each program point, it gives the set of environments

Semantic of programs

Semantic of programs: $\llbracket (L, A) \rrbracket : L \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$

For each program point, it gives the set of environments

It is the smallest solution (in term of inclusion) of the following system:

$$\left\{ \begin{array}{l} R_0 = \mathbb{V} \rightarrow \mathbb{Z} \\ R_{l'} = \bigcup_{(l, c, l') \in A} \llbracket c \rrbracket_C (R_l) \quad l' \neq 0 \end{array} \right.$$

Semantic of programs

Semantic of programs: $\llbracket (L, A) \rrbracket : L \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$

For each program point, it gives the set of environments

It is the smallest solution (in term of inclusion) of the following system:

$$\begin{cases} R_0 = \mathbb{V} \rightarrow \mathbb{Z} \\ R_{l'} = \bigcup_{(l, c, l') \in A} \llbracket c \rrbracket_C (R_l) & l' \neq 0 \end{cases}$$

The theorem of Knaster-Tarski tells us that the solution always exists!

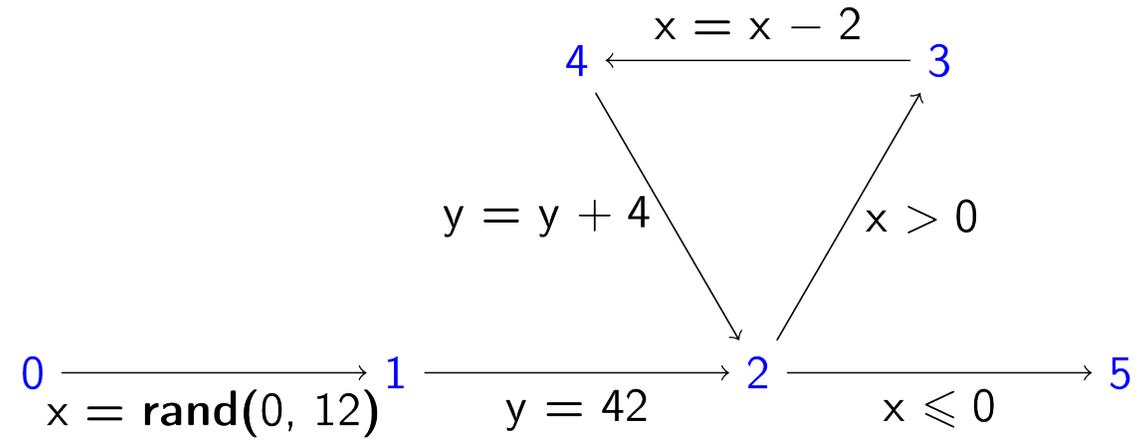
```
0x = rand(0, 12); 1y = 42;
```

```
while 2(x > 0) {
```

```
    3x = x - 2;
```

```
    4y = y + 4;
```

```
}5
```



Equations:

$$R_0 = \{ x \in \mathbb{Z}, y \in \mathbb{Z} \}$$

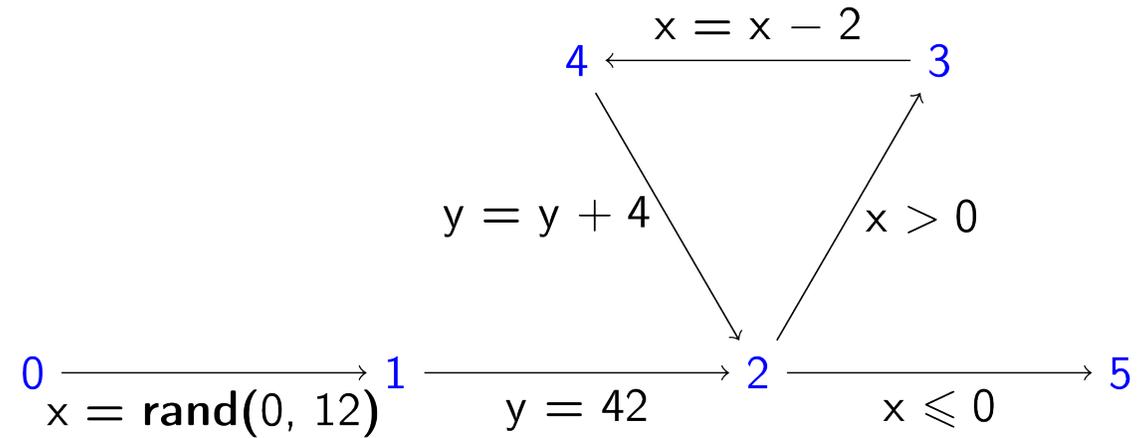
```
0x = rand(0, 12); 1y = 42;
```

```
while 2(x > 0) {
```

```
    3x = x - 2;
```

```
    4y = y + 4;
```

```
}5
```



Equations:

$$R_0 = \{x \in \mathbb{Z}, y \in \mathbb{Z}\}$$

$$R_1 = \{x \in [0, 12], y \in \mathbb{Z}\}$$

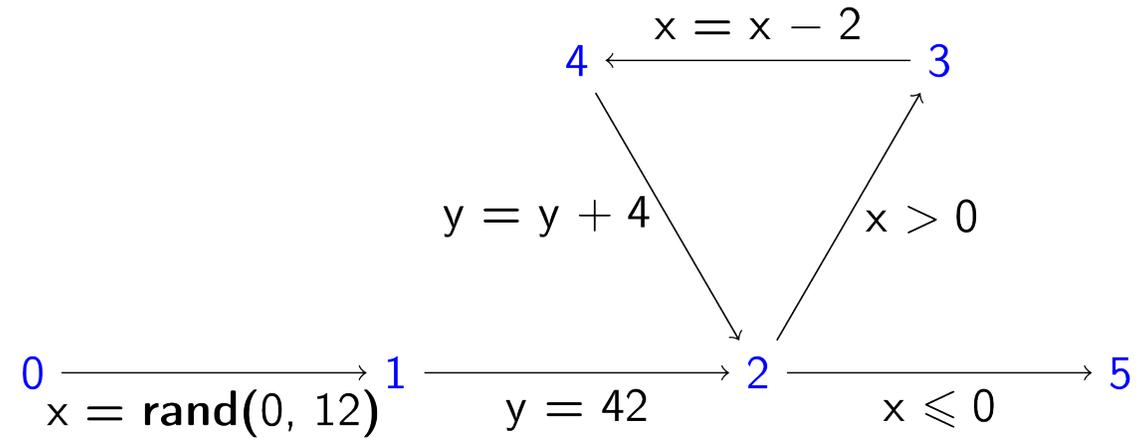
```
0x = rand(0, 12); 1y = 42;
```

```
while 2(x > 0) {
```

```
    3x = x - 2;
```

```
    4y = y + 4;
```

```
}5
```



Equations:

$$R_0 = \{x \in \mathbb{Z}, y \in \mathbb{Z}\}$$

$$R_1 = \{x \in [0, 12], y \in \mathbb{Z}\}$$

$$R_2 = R_1[y \mapsto 42] \cup R_4[y \mapsto y + 4]$$

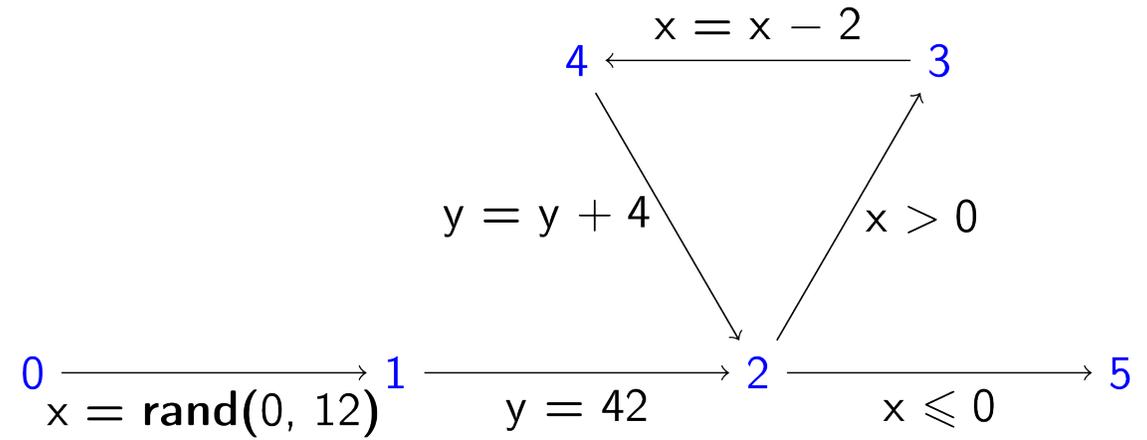
```
0x = rand(0, 12); 1y = 42;
```

```
while 2(x > 0) {
```

```
    3x = x - 2;
```

```
    4y = y + 4;
```

```
}5
```



Equations:

$$R_0 = \{x \in \mathbb{Z}, y \in \mathbb{Z}\}$$

$$R_1 = \{x \in [0, 12], y \in \mathbb{Z}\}$$

$$R_2 = R_1[y \mapsto 42] \cup R_4[y \mapsto y + 4]$$

$$R_3 = R_2 \cap \{x > 0, y \in \mathbb{Z}\}$$

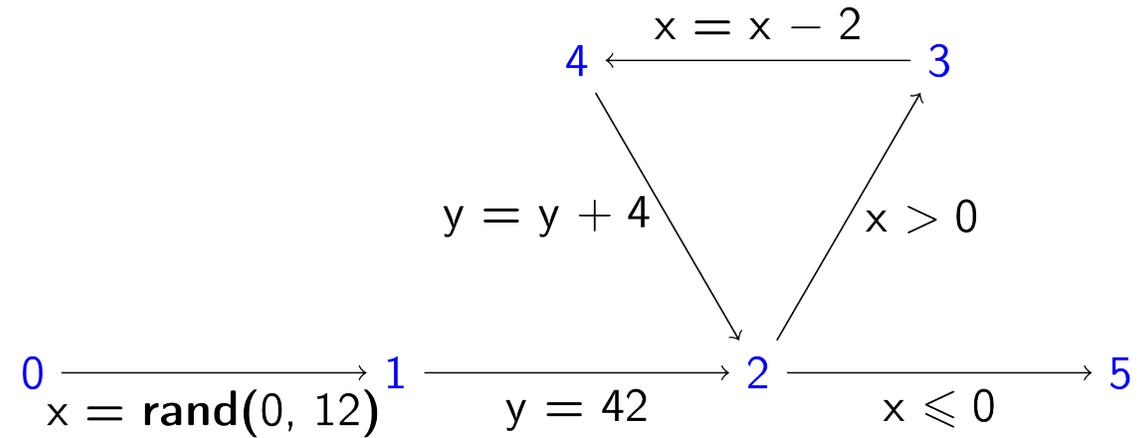
```
0 x = rand(0, 12); 1 y = 42;
```

```
while 2 (x > 0) {
```

```
    3 x = x - 2;
```

```
    4 y = y + 4;
```

```
} 5
```



Equations:

$$R_0 = \{x \in \mathbb{Z}, y \in \mathbb{Z}\}$$

$$R_1 = \{x \in [0, 12], y \in \mathbb{Z}\}$$

$$R_2 = R_1[y \mapsto 42] \cup R_4[y \mapsto y + 4]$$

$$R_3 = R_2 \cap \{x > 0, y \in \mathbb{Z}\}$$

$$R_4 = R_3[x \mapsto x - 2]$$

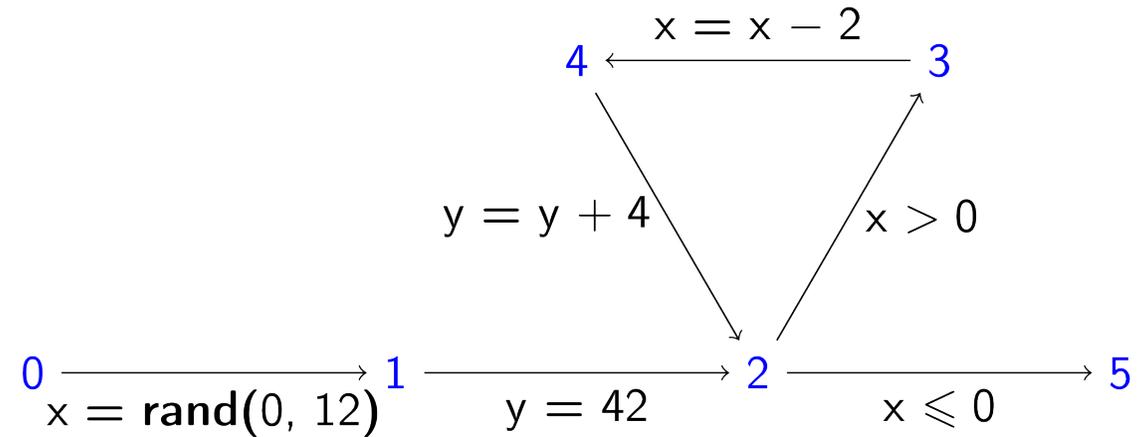
```
0 x = rand(0, 12); 1 y = 42;
```

```
while 2 (x > 0) {
```

```
    3 x = x - 2;
```

```
    4 y = y + 4;
```

```
} 5
```



Equations:

$$R_0 = \{ x \in \mathbb{Z}, y \in \mathbb{Z} \}$$

$$R_1 = \{ x \in [0, 12], y \in \mathbb{Z} \}$$

$$R_2 = R_1[y \mapsto 42] \cup R_4[y \mapsto y + 4]$$

$$R_3 = R_2 \cap \{ x > 0, y \in \mathbb{Z} \}$$

$$R_4 = R_3[x \mapsto x - 2]$$

$$R_5 = R_2 \cap \{ x \leq 0, y \in \mathbb{Z} \}$$

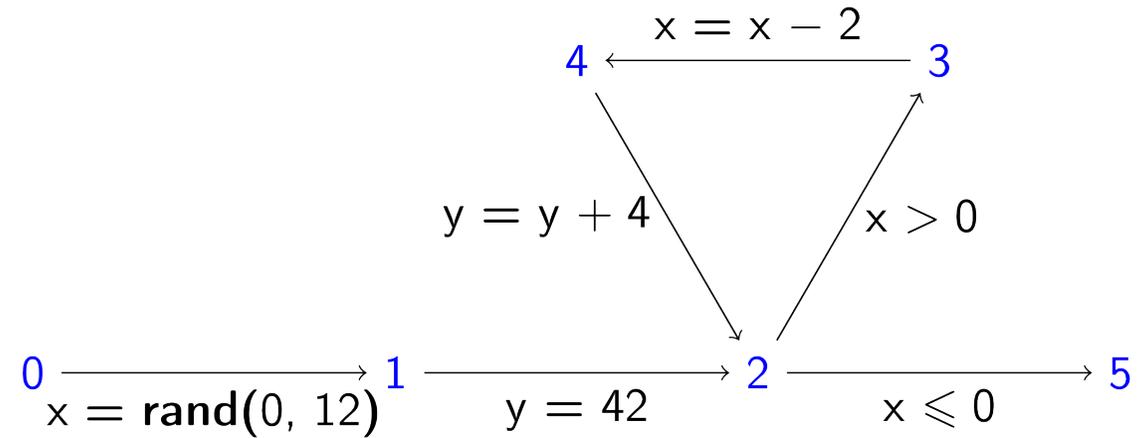
```
0x = rand(0, 12); 1y = 42;
```

```
while 2(x > 0) {
```

```
  3x = x - 2;
```

```
  4y = y + 4;
```

```
}5
```



Equations:

$$R_0 = \{x \in \mathbb{Z}, y \in \mathbb{Z}\}$$

$$R_1 = \{x \in \llbracket 0, 12 \rrbracket, y \in \mathbb{Z}\}$$

$$R_2 = R_1[y \mapsto 42] \cup R_4[y \mapsto y + 4]$$

$$R_3 = R_2 \cap \{x > 0, y \in \mathbb{Z}\}$$

$$R_4 = R_3[x \mapsto x - 2]$$

$$R_5 = R_2 \cap \{x \leq 0, y \in \mathbb{Z}\}$$

Smallest Solution:

$$= \{x \in \mathbb{Z}, y \in \mathbb{Z}\}$$

$$= \{x \in \llbracket 0, 12 \rrbracket, y \in \mathbb{Z}\}$$

$$= \{x \in \llbracket -1, 12 \rrbracket, y \in \llbracket 42, 66 \rrbracket \cap 4\mathbb{Z} + 2 \mid 2x + y \in \llbracket 42, 66 \rrbracket\}$$

$$= \{x \in \llbracket 1, 12 \rrbracket, y \in \llbracket 42, 66 \rrbracket \cap 4\mathbb{Z} + 2 \mid 2x + y \in \llbracket 42, 66 \rrbracket\}$$

$$= \{x \in \llbracket -1, 10 \rrbracket, y \in \llbracket 42, 66 \rrbracket \cap 4\mathbb{Z} + 2 \mid 2x + y \in \llbracket 38, 62 \rrbracket\}$$

$$= \{x \in \llbracket -1, 0 \rrbracket, y \in \llbracket 42, 66 \rrbracket \cap 4\mathbb{Z} + 2 \mid 2x + y \in \llbracket 42, 66 \rrbracket\}$$

Order and Supremum

An **order** \sqsubseteq is a binary relation:

- Reflexive : $\forall x, x \sqsubseteq x$
- Transitive : $\forall x, y, z, (x \sqsubseteq y \wedge y \sqsubseteq z) \Rightarrow x \sqsubseteq z$
- Antisymmetric : $\forall x, y, (x \sqsubseteq y \wedge y \sqsubseteq x) \Rightarrow x = y$

Order and Supremum

An **order** \sqsubseteq is a binary relation:

- Reflexive : $\forall x, x \sqsubseteq x$
- Transitive : $\forall x, y, z, (x \sqsubseteq y \wedge y \sqsubseteq z) \Rightarrow x \sqsubseteq z$
- Antisymmetric : $\forall x, y, (x \sqsubseteq y \wedge y \sqsubseteq x) \Rightarrow x = y$

The **supremum** $\bigsqcup : \mathcal{P}(S) \rightarrow S$ associates to each subset S' of S its smallest upper bound:

- $\forall x \in S', x \sqsubseteq \bigsqcup S'$
- $\forall y \in S, (\forall x \in S', x \sqsubseteq y) \Rightarrow \bigsqcup S' \sqsubseteq y$

Complete Lattice

A set S equipped with an order \sqsubseteq is a ***complete lattice*** if it has a supremum $\bigsqcup : \mathcal{P}(S) \rightarrow S$

Complete Lattice

A set S equipped with an order \sqsubseteq is a **complete lattice** if it has a supremum $\bigsqcup : \mathcal{P}(S) \rightarrow S$

A **complete lattice** automatically has:

- An **infimum** (greatest lower bound):

$$\bigsqcap S' = \bigsqcup \{x \mid \forall y \in S', x \sqsubseteq y\}$$

- A smallest element (**bottom**): $\perp = \bigsqcup \emptyset = \bigsqcap S$

- A greatest element (**top**): $\top = \bigsqcup S = \bigsqcap \emptyset$

Example

\mathbb{Z} is not a complete lattice : $\bigsqcup \mathbb{Z}$ does not exist

Example

\mathbb{Z} is not a complete lattice : $\bigsqcup \mathbb{Z}$ does not exist

$\bar{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, +\infty\}$ is a complete lattice

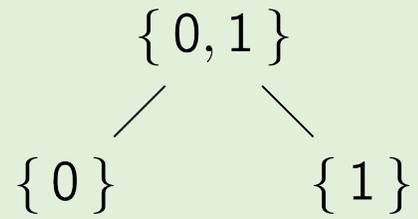
Example

\mathbb{Z} is not a complete lattice : $\bigsqcup \mathbb{Z}$ does not exist

$\bar{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, +\infty\}$ is a complete lattice

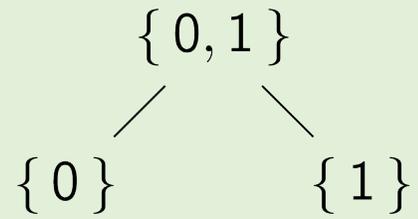
- What is \top ?
- What is \perp ?

Example



Is it a lattice?

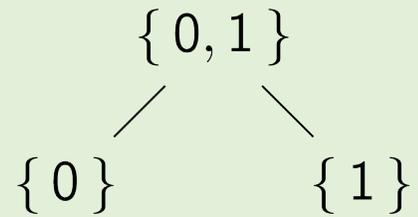
Example



Is it a lattice?

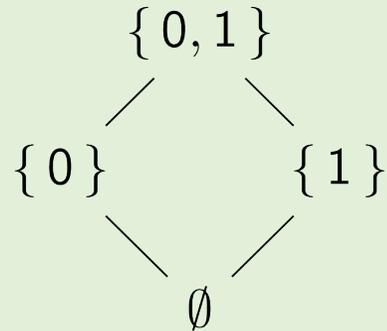
No: $\bigsqcup \emptyset$ does not exist

Example



Is it a lattice?

No: $\sqcup \emptyset$ does not exist



This is a lattice

Monotonic Function

A function f on a complete lattice is *monotonic* if and only if:

$$\forall x, y \in S, \quad x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$

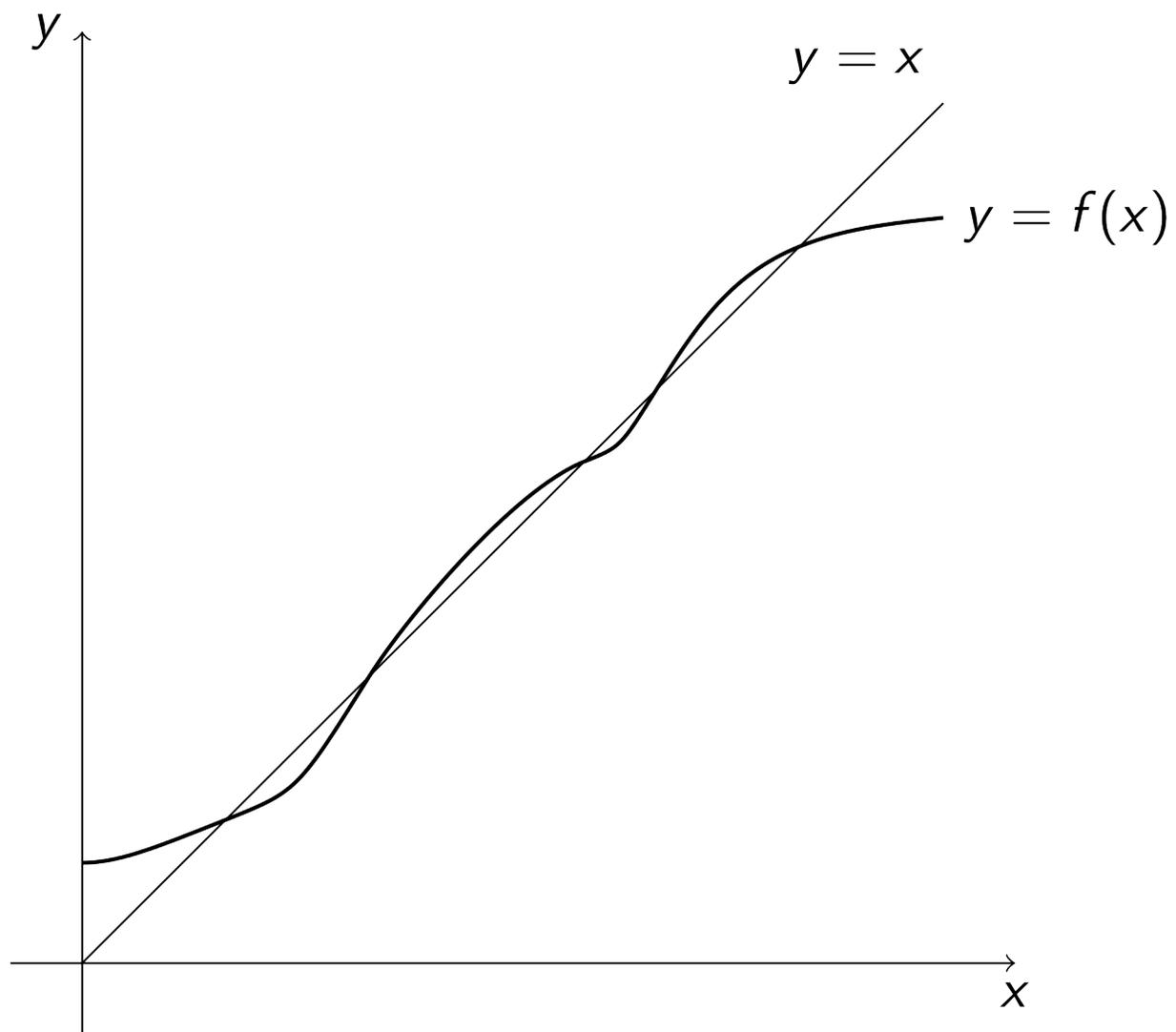
Knaster-Tarski

Theorem of Knaster-Tarski:

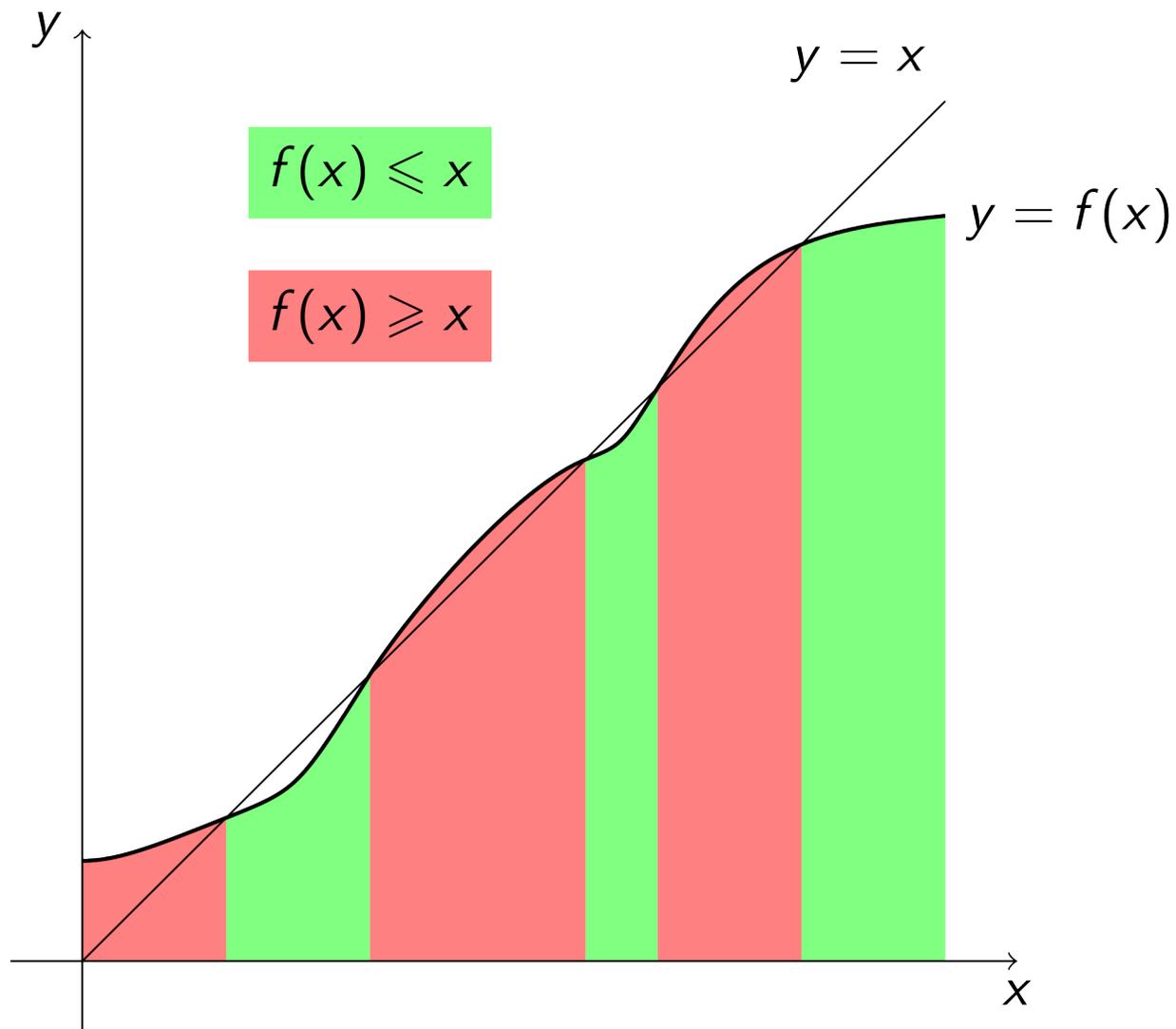
If \mathcal{S} is a complete lattice and f is a monotonic function on this lattice, then f has a ***least fixed point***:

$$\text{lfp } f = \bigcap \{x \in \mathcal{S} \mid f(x) \sqsubseteq x\}$$

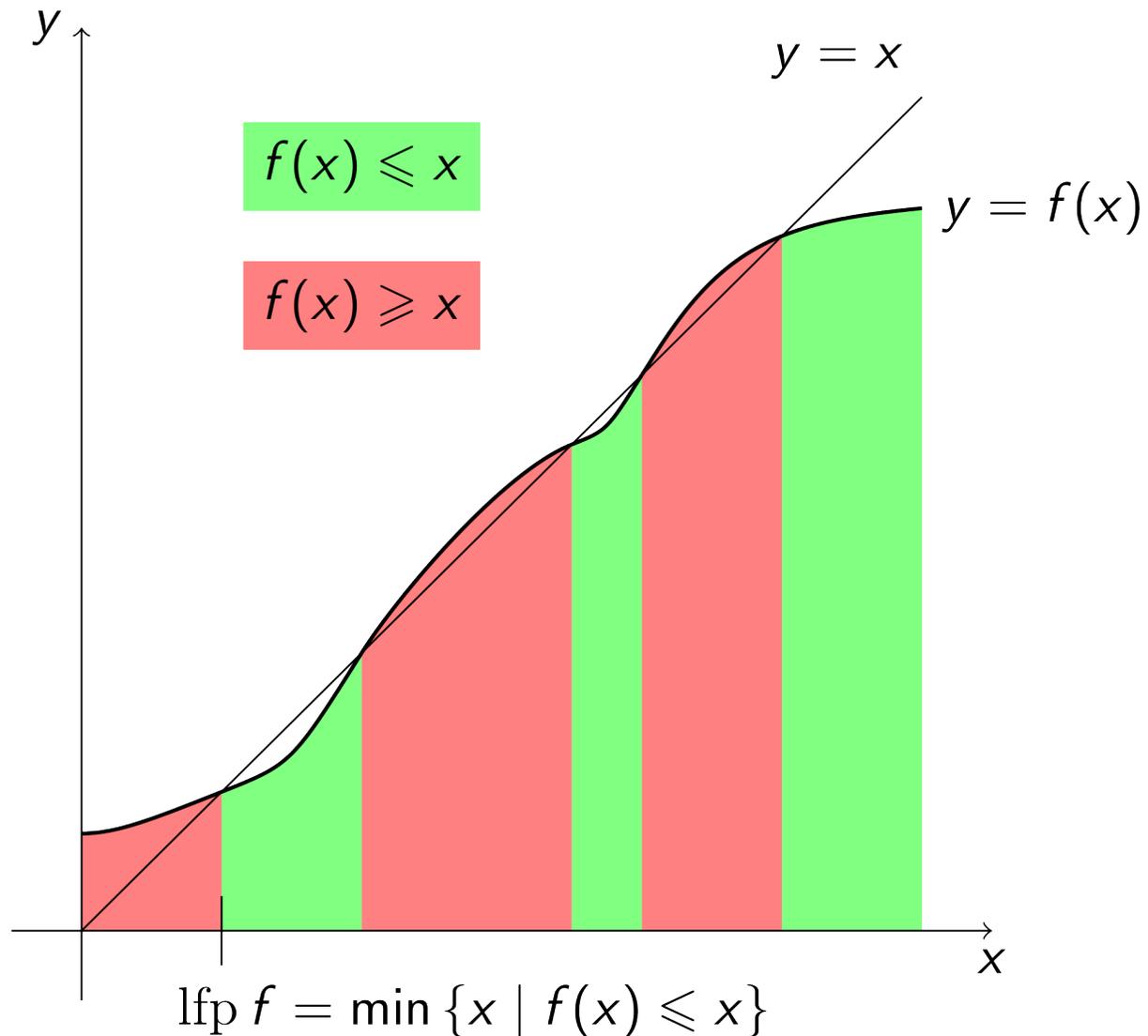
Knaster-Tarski - Idea



Knaster-Tarski - Idea



Knaster-Tarski - Idea



Semantic

- $L \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$ is a complete lattice
- Let $F : (L \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})) \rightarrow (L \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z}))$

$$F(R) = \begin{cases} 0 & \mapsto (\mathbb{V} \rightarrow \mathbb{Z}) \\ I' & \mapsto \bigcup_{(I, c, I') \in A} \llbracket c \rrbracket_C (R(I)) \end{cases}$$

- F is monotonic
- Thus ***lfp*** F exists – Knaster-Tarski

Semantic

- ***lfp* F** is also the smallest solution to our system:

$$\left\{ \begin{array}{l} R_0 = \mathbb{V} \rightarrow \mathbb{Z} \\ R_{l'} = \bigcup_{(l,c,l') \in A} \llbracket c \rrbracket_C (R_l) \end{array} \right. \quad l' \neq 0$$

- Thus our semantic is well defined!

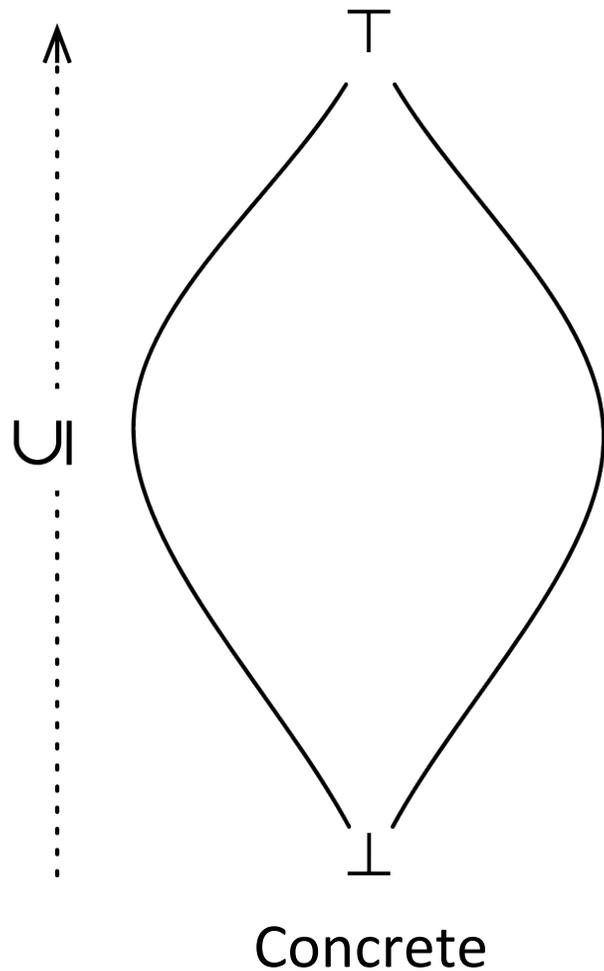
Semantic

- Unfortunately, the concrete semantic cannot be calculated
- We will compute an over-approximation!

Abstract Interpretation

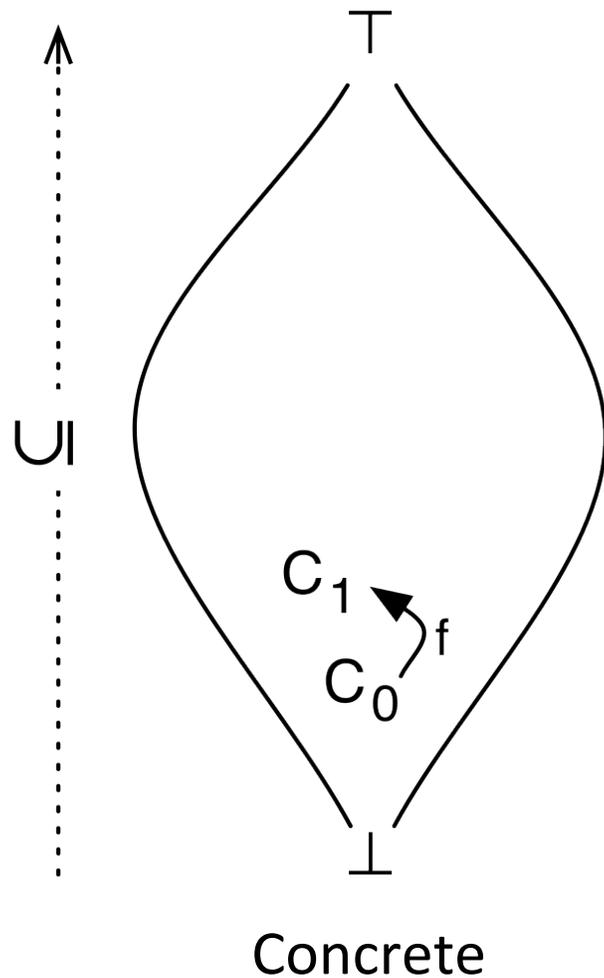
Abstract Interpretation is a constructive theory of sound approximation of fixed points of monotonic functions on complete lattices.

Fixed Point Iteration



Method of computing a fixed point:
 $x, f(x), f(f(x)), f(f(f(x))), \dots$

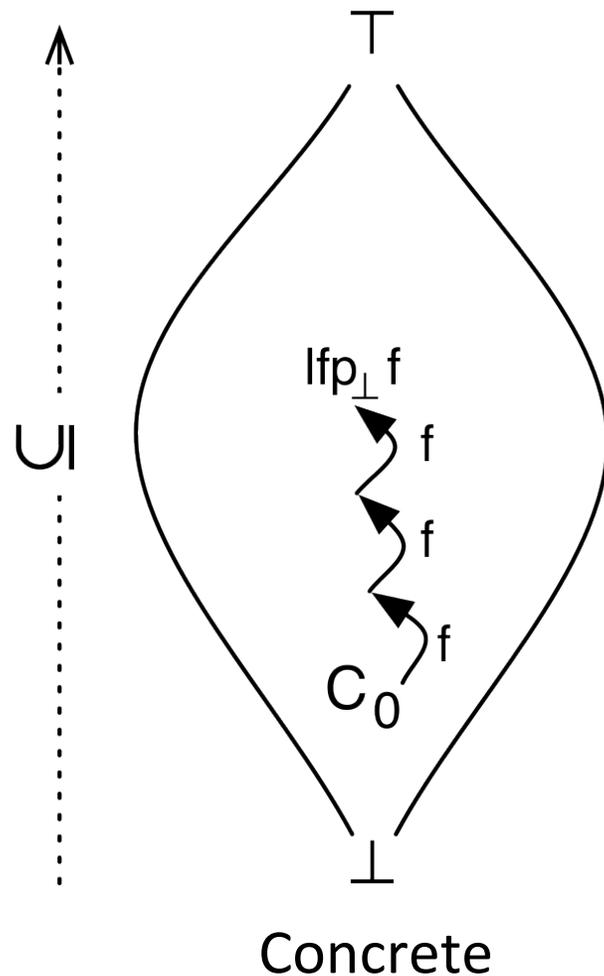
Fixed Point Iteration



Method of computing a fixed point:
 $x, f(x), f(f(x)), f(f(f(x))), \dots$

Start with a point C_0

Fixed Point Iteration

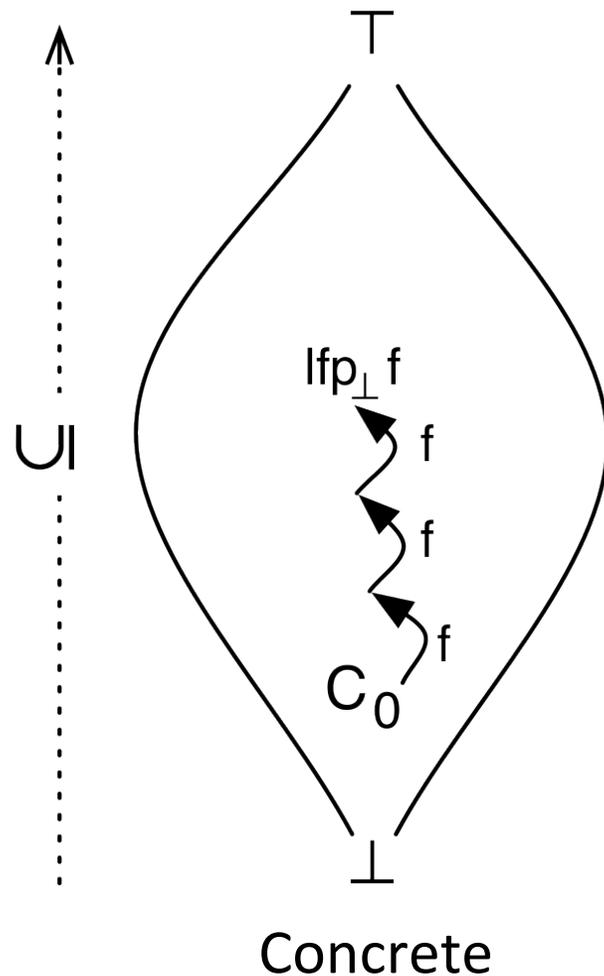


Method of computing a fixed point:
 $x, f(x), f(f(x)), f(f(f(x))), \dots$

Start with a point C_0

$f^n(x) = f^{n+1}(x) \Rightarrow$ **lfp** f found!

Fixed Point Iteration



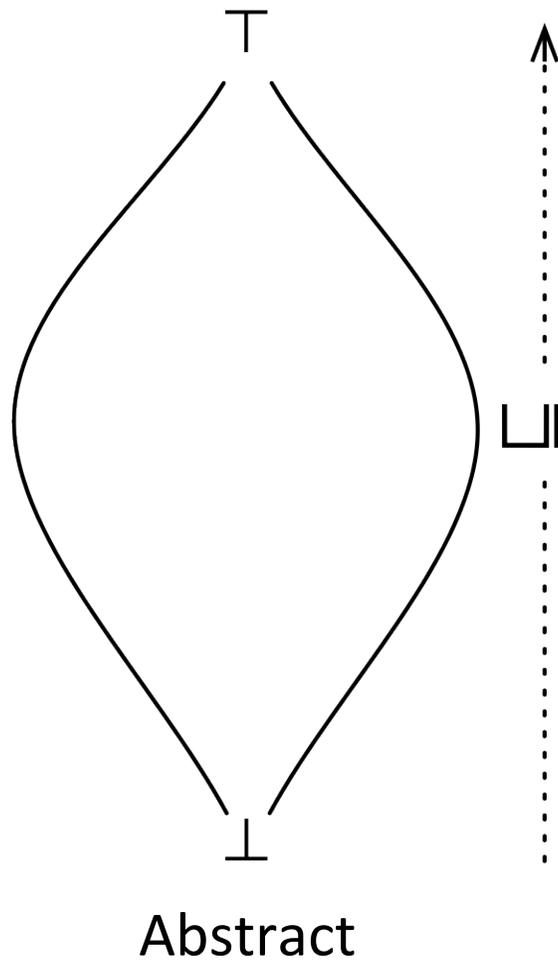
Method of computing a fixed point:
 $x, f(x), f(f(x)), f(f(f(x))), \dots$

Start with a point C_0

$f^n(x) = f^{n+1}(x) \Rightarrow \mathbf{lfp f}$ found!

Problem: Computing $\mathbf{lfp f}$ is **undecidable**

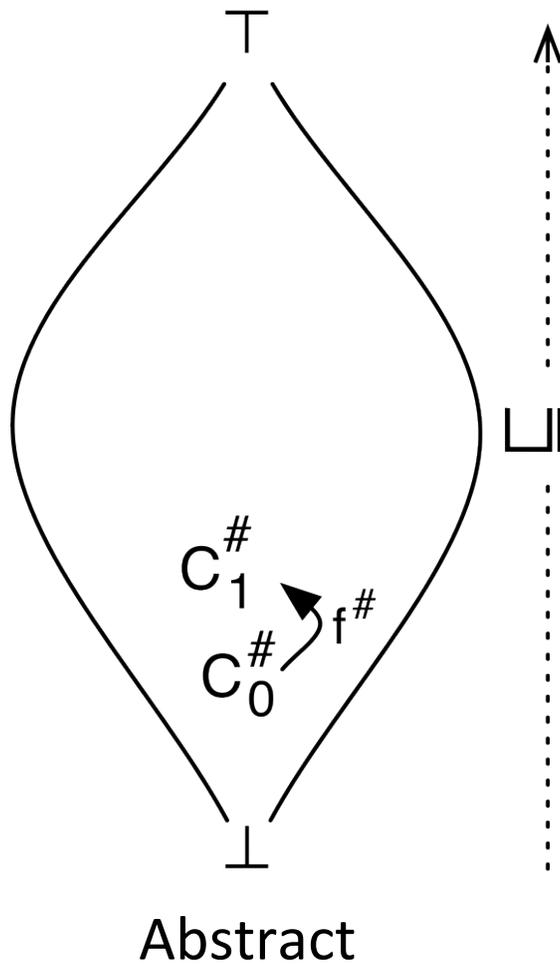
Abstraction



Idea:

- Use a different ***complete lattice***
- Abstract the monotonic function: $f^\#$
- Abstract the entry point: $C_0^\#$

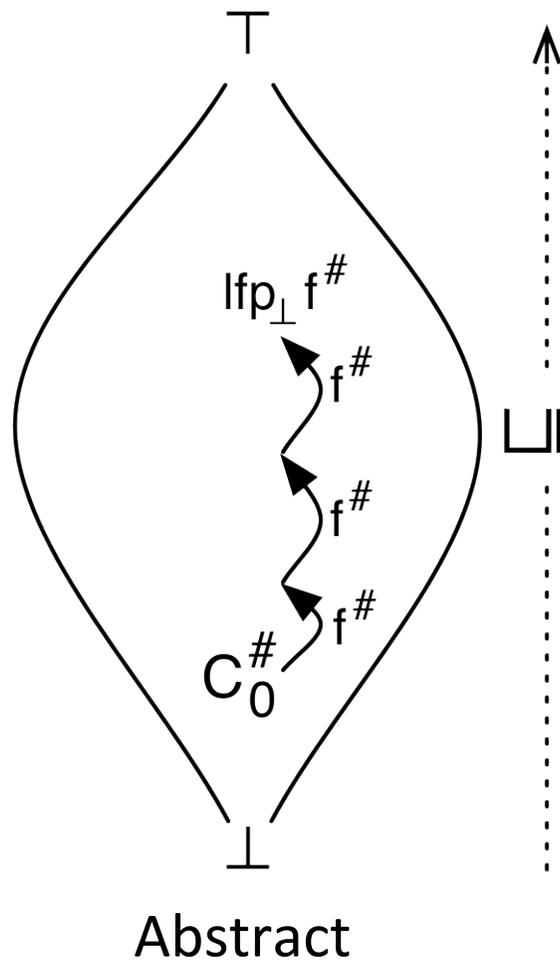
Abstraction



Idea:

- Use a different ***complete lattice***
- Abstract the monotonic function: $f^\#$
- Abstract the entry point: $C_0^\#$

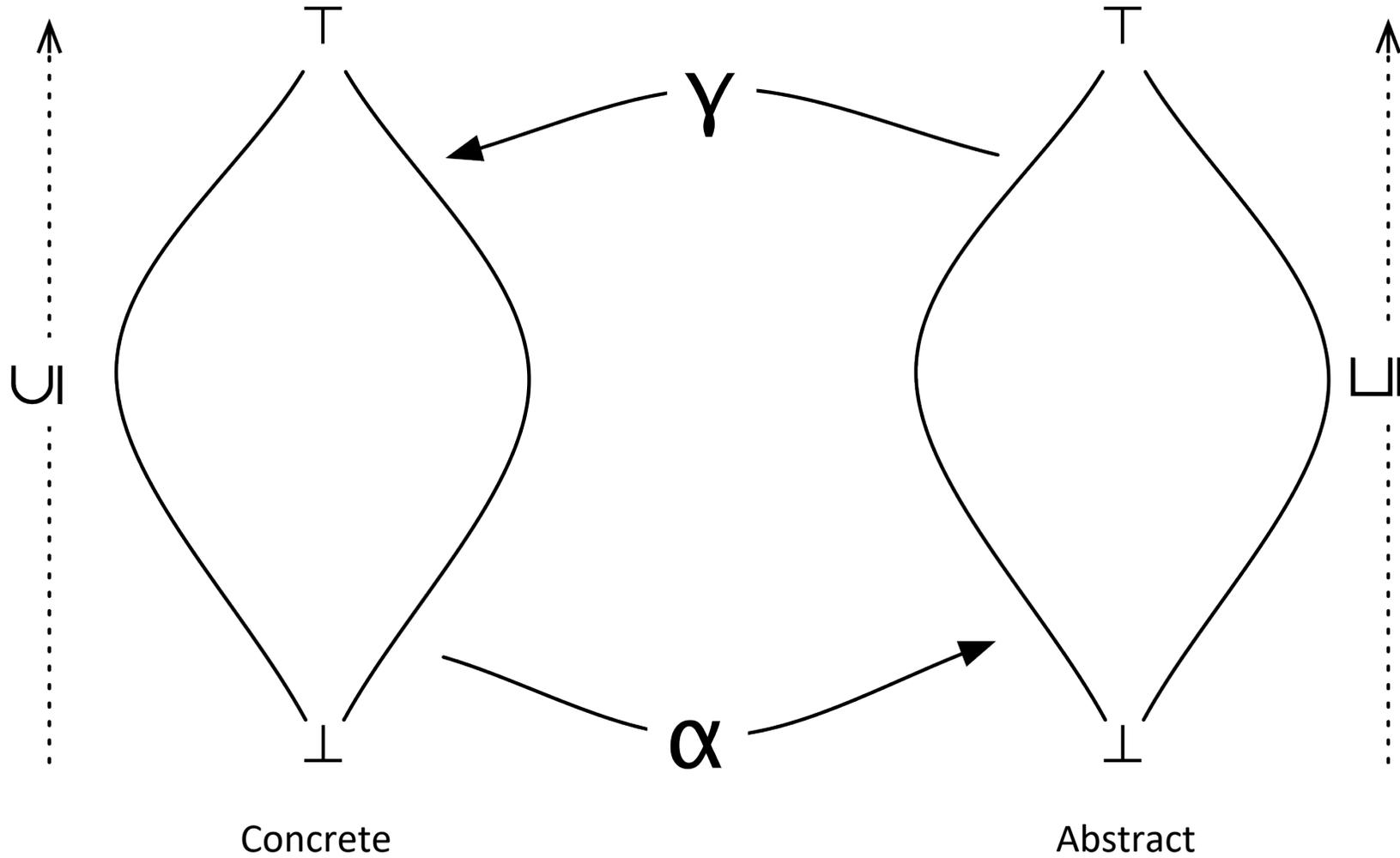
Abstraction



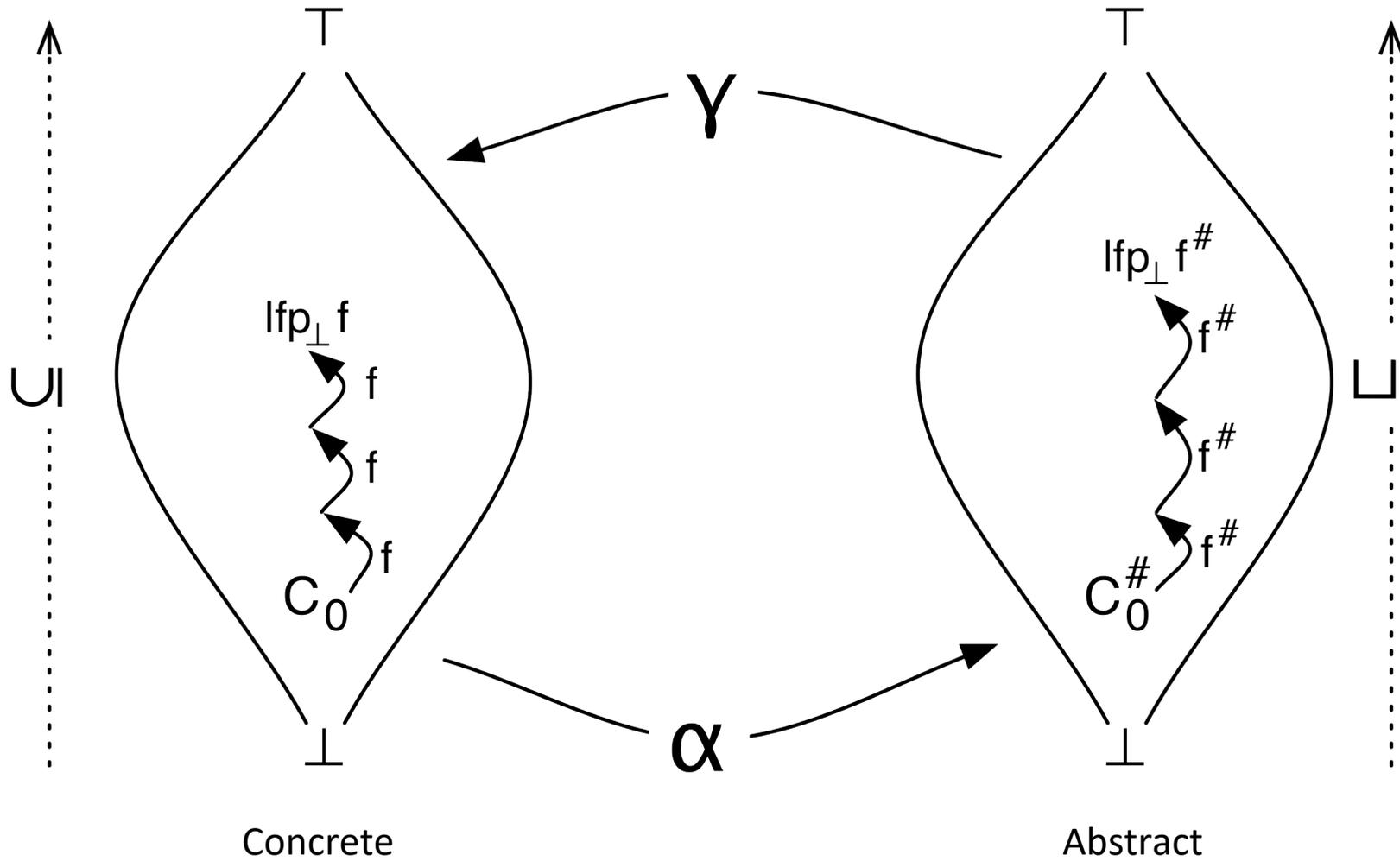
Idea:

- Use a different ***complete lattice***
- Abstract the monotonic function: $f^\#$
- Abstract the entry point: $C_0^\#$

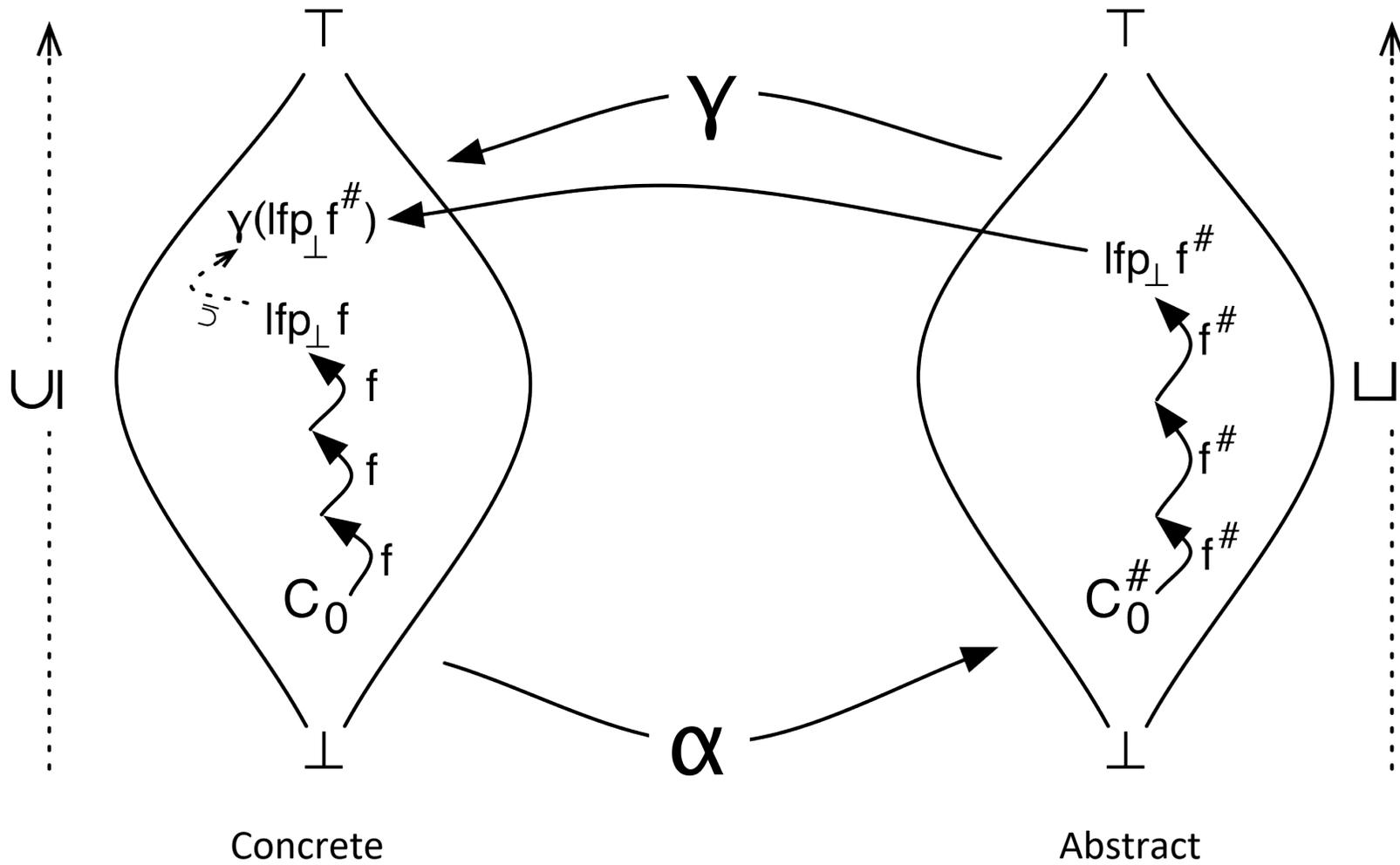
Galois Connection



Galois Connection



Galois Connection



Build an abstraction

- Goal: Abstract $L \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$

Build an abstraction

- Goal: Abstract $L \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$
- L : finite set of program points – keep

Build an abstraction

- Goal: Abstract $L \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$
- L : finite set of program points – keep
- \mathbb{V} : finite set of variables – keep

Build an abstraction

- Goal: Abstract $L \rightarrow \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$
- L : finite set of program points – keep
- \mathbb{V} : finite set of variables – keep
- \mathbb{Z} : infinite set of integers – abstract!

Build an abstraction

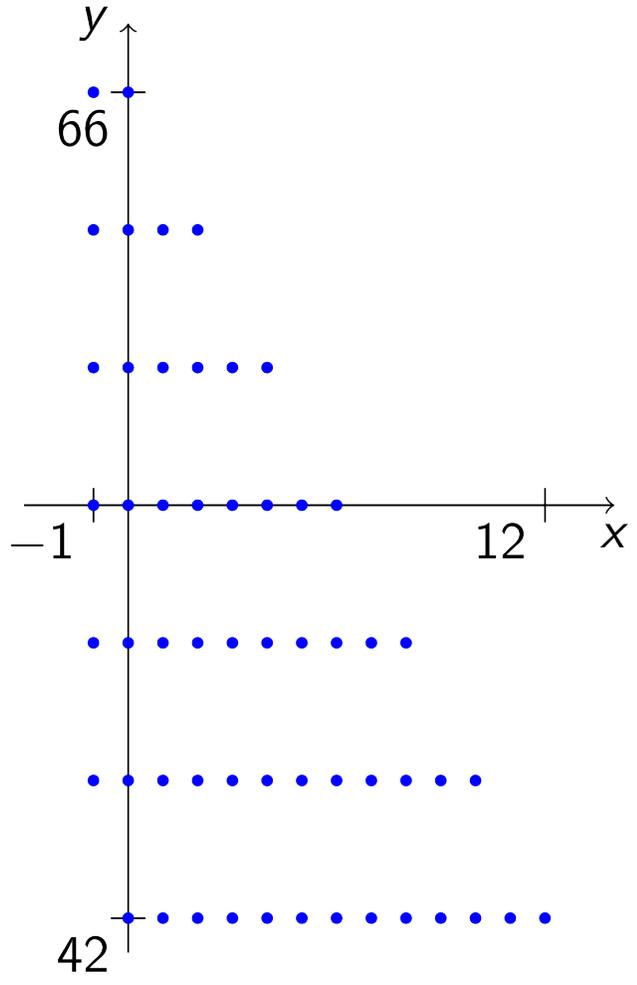
- Goal: Abstract $\mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$

Build an abstraction

- Goal: Abstract $\mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$
 - Non-relational : $\mathbb{V} \rightarrow \mathcal{P}(\mathbb{Z})$ then $\mathbb{V} \rightarrow \mathcal{D}^\#$

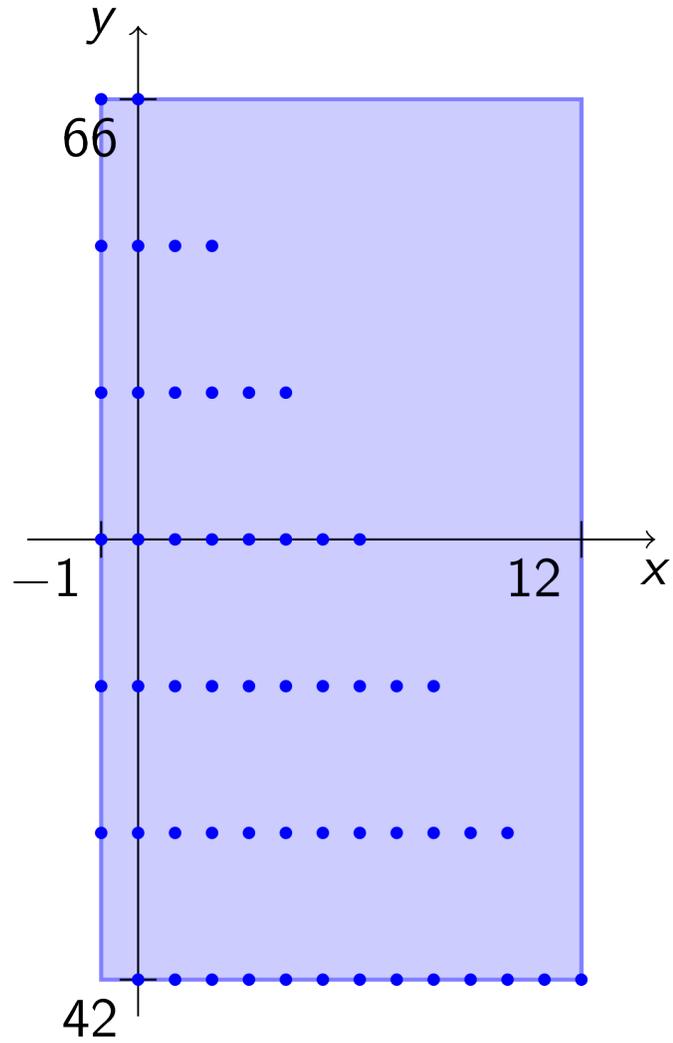
Build an abstraction

- Goal: Abstract $\mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$
 - Non-relational : $\mathbb{V} \rightarrow \mathcal{P}(\mathbb{Z})$ then $\mathbb{V} \rightarrow \mathcal{D}^\#$
 - Relational : $\mathcal{D}^\#$

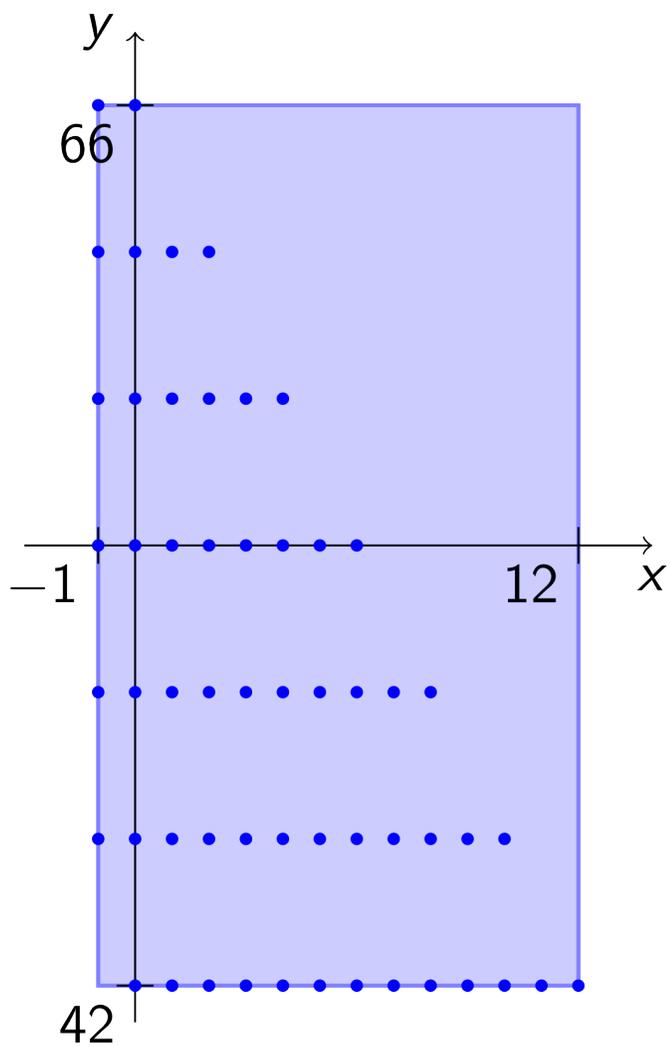


Reachable states at program point 2

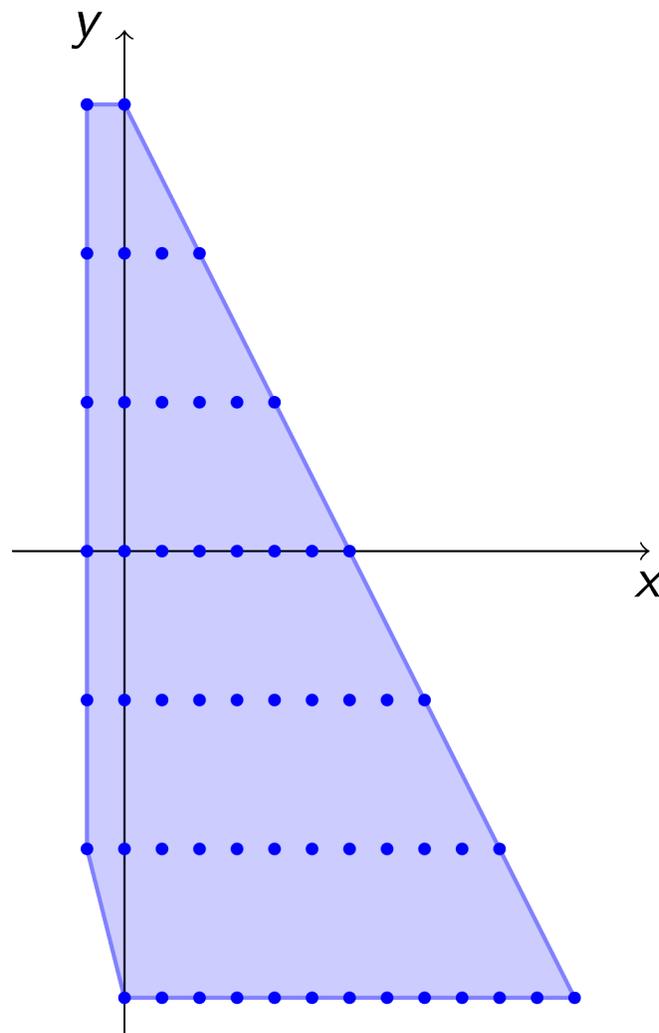
Non-relational abstraction



Non-relational
abstraction

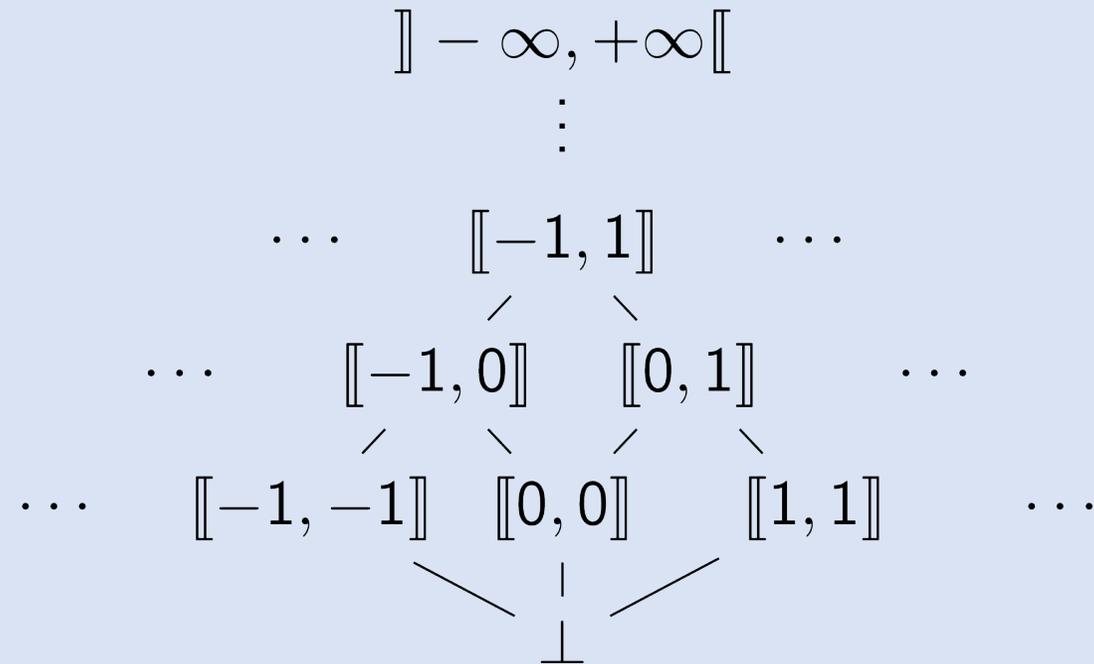


Relational
abstraction



Interval domain

Interval lattice $(\mathcal{D}^\#, \sqsubseteq^\#)$:



We first need to make sure it is a **complete lattice**

Interval domain

Concretization function:

$$\gamma(\llbracket -\infty, +\infty \rrbracket) = \llbracket -\infty, +\infty \llbracket$$

$$\gamma(\llbracket -\infty, n \rrbracket) = \llbracket -\infty, n \rrbracket$$

$$\gamma(\llbracket n, +\infty \rrbracket) = \llbracket n, +\infty \llbracket$$

$$\gamma(\llbracket n_1, n_2 \rrbracket) = \llbracket n_1, n_2 \rrbracket$$

$$\gamma(\perp) = \emptyset$$

Interval domain

Abstraction function:

$$\alpha(S) = \begin{cases} \llbracket n_1, n_2 \rrbracket & \text{with } n_1 = \min S \text{ and } n_2 = \max S \\ \perp & \text{if } S = \emptyset \end{cases}$$

Interval domain

(α, γ) is a Galois connection

Semantic of Expressions

Semantic of expressions: $\llbracket e \rrbracket_E^\# : (\mathbb{V} \rightarrow \mathcal{D}^\#) \rightarrow \mathcal{D}^\#$

Semantic of Expressions

Semantic of expressions: $\llbracket e \rrbracket_{\mathbb{E}}^{\#} : (\mathbb{V} \rightarrow \mathcal{D}^{\#}) \rightarrow \mathcal{D}^{\#}$

$$\llbracket v \rrbracket_{\mathbb{E}}^{\#} (\rho) = \rho(v)$$

$$\llbracket n \rrbracket_{\mathbb{E}}^{\#} (\rho) = n^{\#}$$

$$\llbracket \mathbf{rand}(n_1, n_2) \rrbracket_{\mathbb{E}}^{\#} (\rho) = \mathbf{rand}^{\#}(n_1, n_2)$$

$$\llbracket e_1 + e_2 \rrbracket_{\mathbb{E}}^{\#} (\rho) = \llbracket e_1 \rrbracket_{\mathbb{E}}^{\#} +^{\#} \llbracket e_2 \rrbracket_{\mathbb{E}}^{\#}$$

...

Semantic of Expressions

Abstract operators:

$$n^\# = \alpha(\{ n \}) = \llbracket n, n \rrbracket$$

Semantic of Expressions

Abstract operators:

$$n^\# = \alpha(\{n\}) = \llbracket n, n \rrbracket$$

$$\mathbf{rand}^\#(n_1, n_2) = \alpha(\llbracket n_1, n_2 \rrbracket) = \begin{cases} \llbracket n_1, n_2 \rrbracket & \text{si } n_1 \leq n_2 \\ \perp & \text{si } n_1 > n_2 \end{cases}$$

Semantic of Expressions

Abstract operators:

$$n^\# = \alpha(\{n\}) = \llbracket n, n \rrbracket$$

$$\mathbf{rand}^\#(n_1, n_2) = \alpha(\llbracket n_1, n_2 \rrbracket) = \begin{cases} \llbracket n_1, n_2 \rrbracket & \text{si } n_1 \leq n_2 \\ \perp & \text{si } n_1 > n_2 \end{cases}$$

$$x^\# +^\# y^\# = \alpha\left(\left\{x + y \mid x \in \gamma(x^\#), y \in \gamma(y^\#)\right\}\right) = \begin{cases} \llbracket a + c, b + d \rrbracket & \text{avec } x^\# = \llbracket a, b \rrbracket \text{ et } y^\# = \llbracket c, d \rrbracket \\ \perp & \text{si } x^\# = \perp \text{ ou } y^\# = \perp \end{cases}$$

...

Semantic of Expressions

Abstract operators:

$$x^\# -^\# y^\# = \alpha \left(\left\{ x - y \mid x \in \gamma(x^\#), y \in \gamma(y^\#) \right\} \right) =$$
$$\begin{cases} \llbracket a - d, b - c \rrbracket & \text{avec } x^\# = \llbracket a, b \rrbracket \text{ et } y^\# = \llbracket c, d \rrbracket \\ \perp & \text{si } x^\# = \perp \text{ ou } y^\# = \perp \end{cases}$$

Semantic of Expressions

Abstract operators:

$$x^\# -^\# y^\# = \alpha \left(\left\{ x - y \mid x \in \gamma(x^\#), y \in \gamma(y^\#) \right\} \right) =$$

$$\begin{cases} \llbracket a - d, b - c \rrbracket & \text{avec } x^\# = \llbracket a, b \rrbracket \text{ et } y^\# = \llbracket c, d \rrbracket \\ \perp & \text{si } x^\# = \perp \text{ ou } y^\# = \perp \end{cases}$$

$$x^\# \times^\# y^\# = \alpha \left(\left\{ x \times y \mid x \in \gamma(x^\#), y \in \gamma(y^\#) \right\} \right) =$$

$$\begin{cases} \llbracket \min(ab, ac, ad, bd), \max(ab, ac, ad, bd) \rrbracket & \text{avec } \dots \\ \perp & \text{si } \dots \end{cases}$$

Semantic of commands

Semantic of commands: $\llbracket c \rrbracket_C^\# : (\mathbb{V} \rightarrow \mathcal{D}^\#) \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#)$

Semantic of commands

Semantic of commands: $\llbracket c \rrbracket_C^\# : (\mathbb{V} \rightarrow \mathcal{D}^\#) \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#)$

$$\llbracket v = e \rrbracket_C^\# (\rho) = \rho \left[v \mapsto \llbracket e \rrbracket_E^\# \rho \right]$$

$$\llbracket e > 0 \rrbracket_C^\# (\rho) = \begin{cases} \rho \left[v \mapsto \rho(v) \sqcap^\# \alpha(\llbracket 1, +\infty \rrbracket) \right] & \text{si } e = v \\ \rho & \text{sinon} \end{cases}$$

Semantic of programs

Semantic of programs: $\llbracket (L, A) \rrbracket^\# : L \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#)$

Semantic of programs

Semantic of programs: $\llbracket (L, A) \rrbracket^\# : L \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#)$

It is the smallest solution (in term of inclusion) of the following system:

$$\left\{ \begin{array}{l} R_0^\# = \mathbb{V} \rightarrow \top \\ R_{l'}^\# = \bigsqcup_{nr}^\# \llbracket c \rrbracket_C^\# (R_l^\#) \quad l' \neq 0 \\ (l, c, l') \in A \end{array} \right.$$

Semantic of programs

Semantic of programs: $\llbracket (L, A) \rrbracket^\# : L \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#)$

It is the smallest solution (in term of inclusion) of the following system:

$$\left\{ \begin{array}{l} R_0^\# = \mathbb{V} \rightarrow \top \\ R_{l'}^\# = \bigsqcup_{nr}^\# \llbracket c \rrbracket_C^\# (R_l^\#) \quad l' \neq 0 \\ (l, c, l') \in A \end{array} \right.$$

Knaster-Tarski: the solution exists!

Computing the fixpoint

- We define $F^\# : (L \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#)) \rightarrow (L \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#))$

$$F^\#(R^\#) = \begin{cases} 0 & \mapsto \top_{\text{nr}} \\ l' & \mapsto \bigsqcup_{(l,c,l') \in A} \llbracket c \rrbracket_C^\# (R^\#(l)) \end{cases}$$

Computing the fixpoint

- We define $F^\# : (L \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#)) \rightarrow (L \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#))$

$$F^\#(R^\#) = \begin{cases} 0 & \mapsto \top_{\text{nr}} \\ l' & \mapsto \bigsqcup_{(l,c,l') \in A} \llbracket c \rrbracket_C^\# (R^\#(l)) \end{cases}$$

- $F^\#$ is monotonic and computable

Computing the fixpoint

- We define $F^\# : (L \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#)) \rightarrow (L \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#))$

$$F^\#(R^\#) = \begin{cases} 0 & \mapsto \top_{\text{nr}} \\ l' & \mapsto \bigsqcup_{(l,c,l') \in A}^\# \llbracket c \rrbracket_C^\# (R^\#(l)) \end{cases}$$

- $F^\#$ is monotonic and computable
- $\text{lfp } F^\#$ is the abstract semantic of our program

Computing the fixpoint

- We define $F^\# : (L \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#)) \rightarrow (L \rightarrow (\mathbb{V} \rightarrow \mathcal{D}^\#))$

$$F^\#(R^\#) = \begin{cases} 0 & \mapsto \top_{\text{nr}} \\ l' & \mapsto \bigsqcup_{(l,c,l') \in A}^\# \llbracket c \rrbracket_C^\#(R^\#(l)) \end{cases}$$

- $F^\#$ is monotonic and computable
- lfp $F^\#$ is the abstract semantic of our program
- Iterative method to compute the least fixed point:
 - $R^{\#0} := L \rightarrow \perp_{\text{nr}}$
 - $R^{\#k+1} := F^\#(R^{\#k})$
 - Stop when $R^{\#k+1} = R^{\#k}$

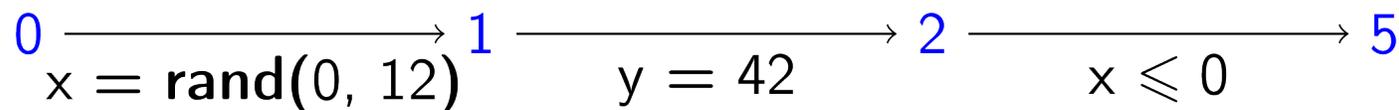
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

}5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$$

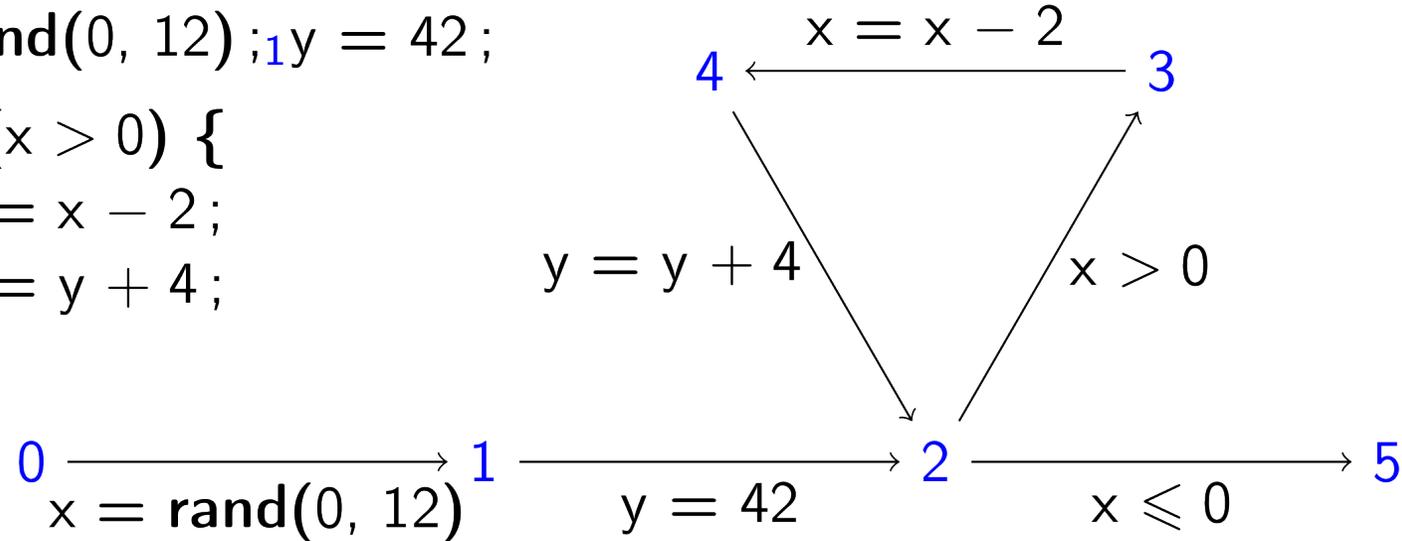
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)		
1	(\perp, \perp)		
2	(\perp, \perp)		
3	(\perp, \perp)		
4	(\perp, \perp)		
5	(\perp, \perp)		

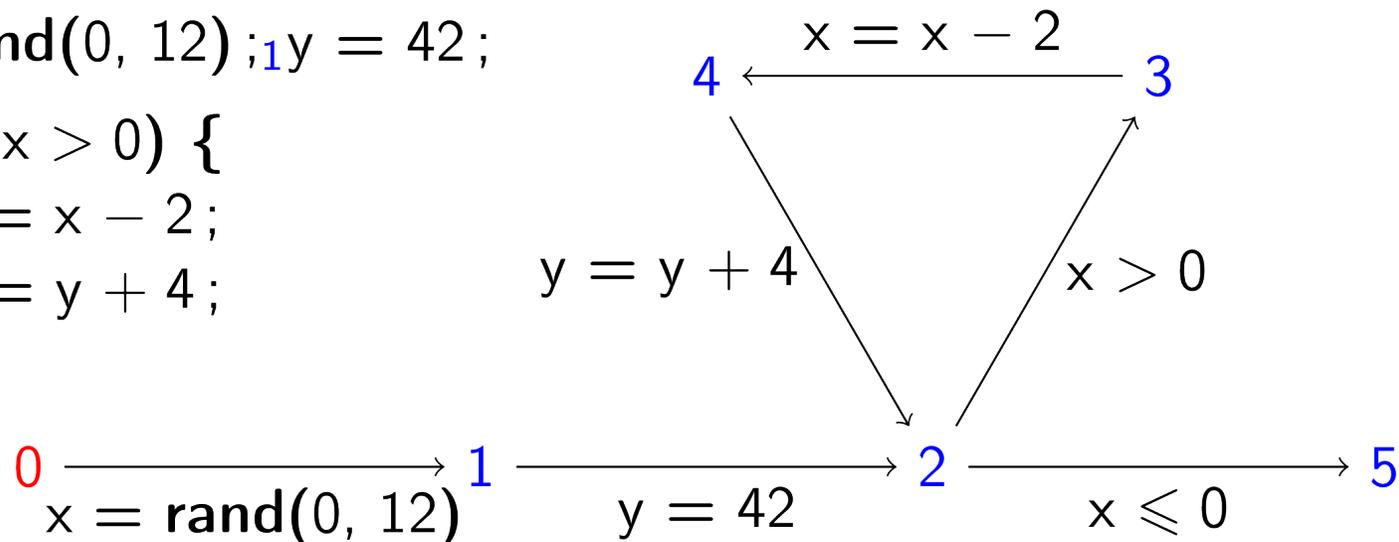
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

}5



$R_0^{\#i+1} = \top$

$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$

$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$

$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$

$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$

$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$

$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$

$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$

$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$

l	$R_l^{\#0}$	$R_l^{\#1}$	$R_l^{\#2}$
0	(\perp, \perp)	(\top, \top)	
1	(\perp, \perp)		
2	(\perp, \perp)		
3	(\perp, \perp)		
4	(\perp, \perp)		
5	(\perp, \perp)		

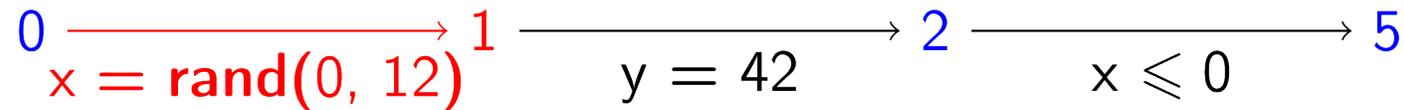
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$$

l	$R_l^{\#0}$	$R_l^{\#1}$	$R_l^{\#2}$
0	(\perp, \perp)	(\top, \top)	
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	
2	(\perp, \perp)		
3	(\perp, \perp)		
4	(\perp, \perp)		
5	(\perp, \perp)		

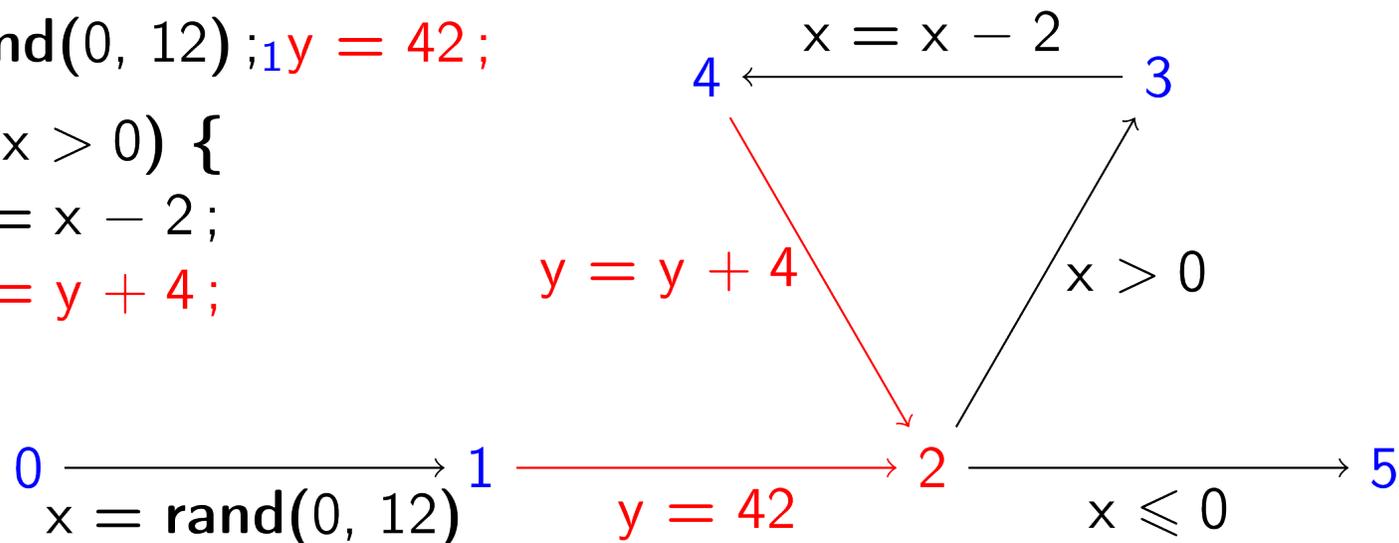
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{nr}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)	(\top, \top)	
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	
2	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \{42\})$	
3	(\perp, \perp)		
4	(\perp, \perp)		
5	(\perp, \perp)		

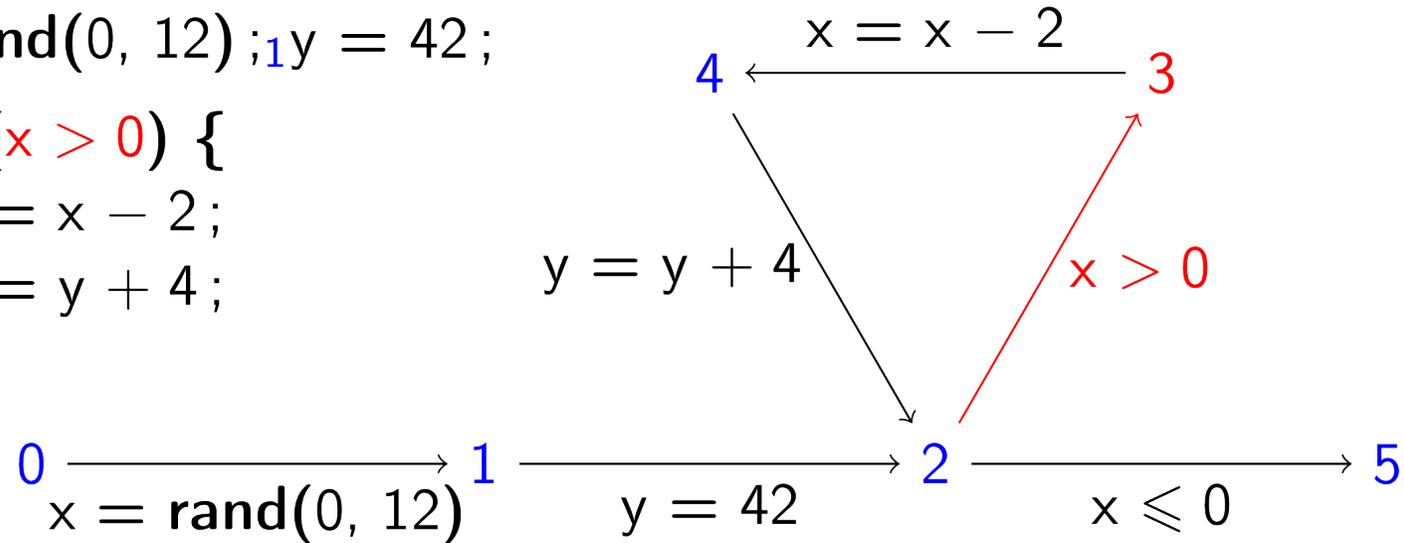
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\cap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\cap^{\#} \llbracket -\infty, 0 \rrbracket]$$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)	(\top, \top)	
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	
2	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \{42\})$	
3	(\perp, \perp)	$(\llbracket 1, 12 \rrbracket, \{42\})$	
4	(\perp, \perp)		
5	(\perp, \perp)		

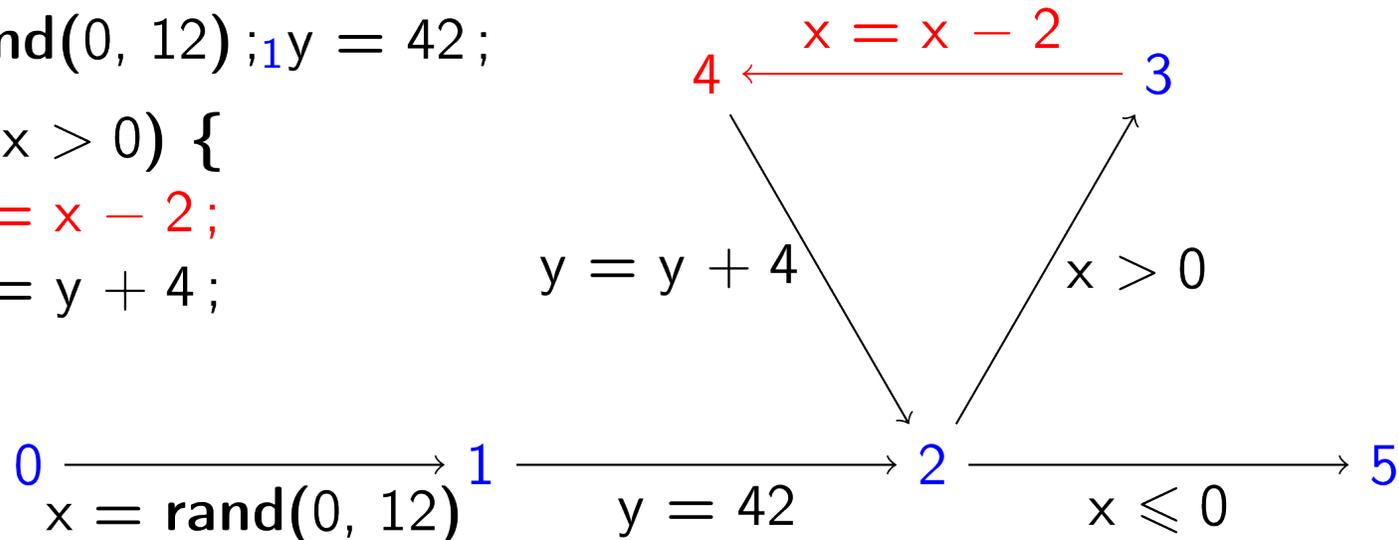
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)	(\top, \top)	
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	
2	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \{42\})$	
3	(\perp, \perp)	$(\llbracket 1, 12 \rrbracket, \{42\})$	
4	(\perp, \perp)	$(\llbracket -1, 10 \rrbracket, \{42\})$	
5	(\perp, \perp)		

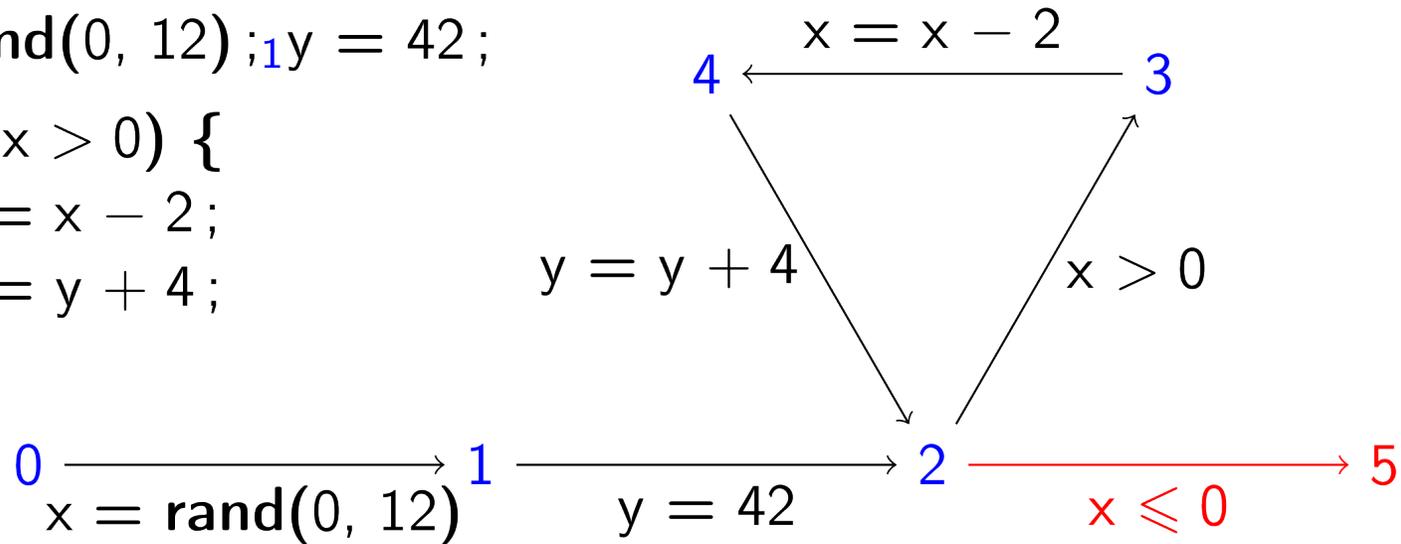
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)	(\top, \top)	
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	
2	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \{42\})$	
3	(\perp, \perp)	$(\llbracket 1, 12 \rrbracket, \{42\})$	
4	(\perp, \perp)	$(\llbracket -1, 10 \rrbracket, \{42\})$	
5	(\perp, \perp)	$(\{0\}, \{42\})$	

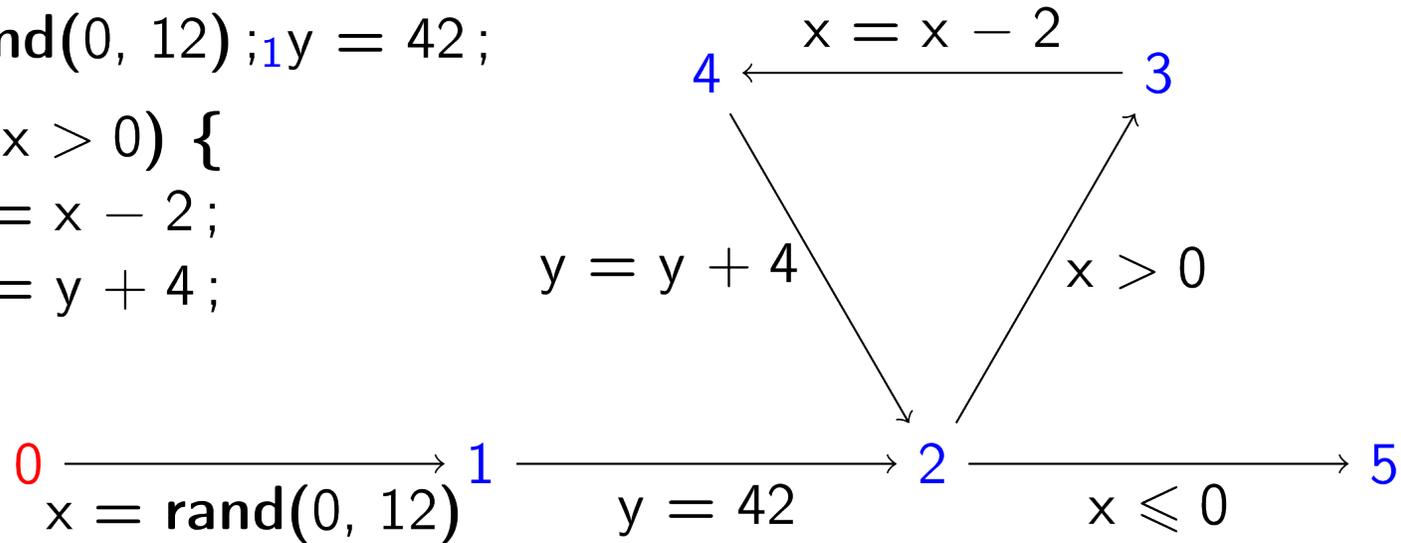
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

}5



$R_0^{\#i+1} = \top$

$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$

$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$

$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$

$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$

$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$

$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$

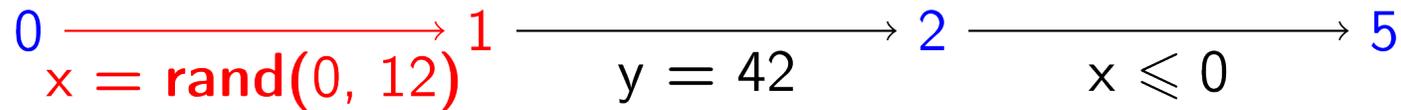
$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$

$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)	(\top, \top)	(\top, \top)
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	
2	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \{42\})$	
3	(\perp, \perp)	$(\llbracket 1, 12 \rrbracket, \{42\})$	
4	(\perp, \perp)	$(\llbracket -1, 10 \rrbracket, \{42\})$	
5	(\perp, \perp)	$(\{0\}, \{42\})$	

$0x = \text{rand}(0, 12); 1y = 42;$

while $2(x > 0)$ {
 $3x = x - 2;$
 $4y = y + 4;$
} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#} R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x) \sqcap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x) \sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)	(\top, \top)	(\top, \top)
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	$(\llbracket 0, 12 \rrbracket, \top)$
2	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \{42\})$	
3	(\perp, \perp)	$(\llbracket 1, 12 \rrbracket, \{42\})$	
4	(\perp, \perp)	$(\llbracket -1, 10 \rrbracket, \{42\})$	
5	(\perp, \perp)	$(\{0\}, \{42\})$	

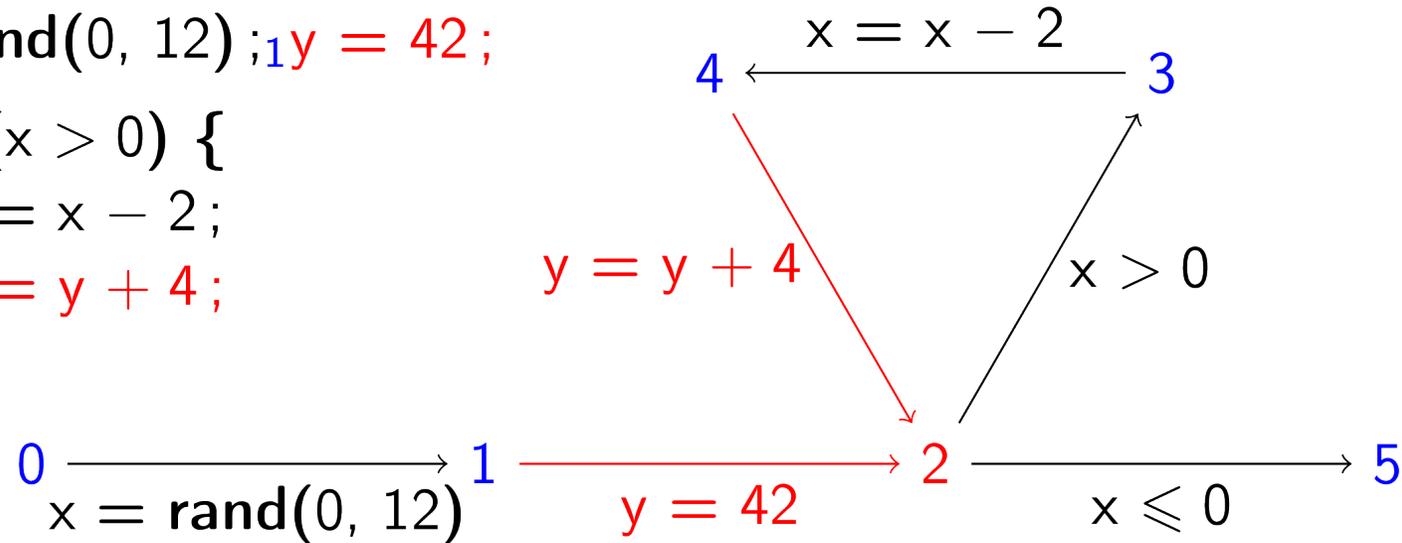
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{nr}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} \left[x \mapsto R_2^{\#i+1}(x) \right. \\ \left. \cap^{\#} \llbracket 1, +\infty \rrbracket \right]$$

$$R_4^{\#i+1} = R_3^{\#i+1} \left[x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket \right]$$

$$R_5^{\#i+1} = R_2^{\#i+1} \left[x \mapsto R_2^{\#i+1}(x) \right. \\ \left. \cap^{\#} \llbracket -\infty, 0 \rrbracket \right]$$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)	(\top, \top)	(\top, \top)
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	$(\llbracket 0, 12 \rrbracket, \top)$
2	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \{42\})$	$(\llbracket -1, 12 \rrbracket, \llbracket 42, 46 \rrbracket)$
3	(\perp, \perp)	$(\llbracket 1, 12 \rrbracket, \{42\})$	
4	(\perp, \perp)	$(\llbracket -1, 10 \rrbracket, \{42\})$	
5	(\perp, \perp)	$(\{0\}, \{42\})$	

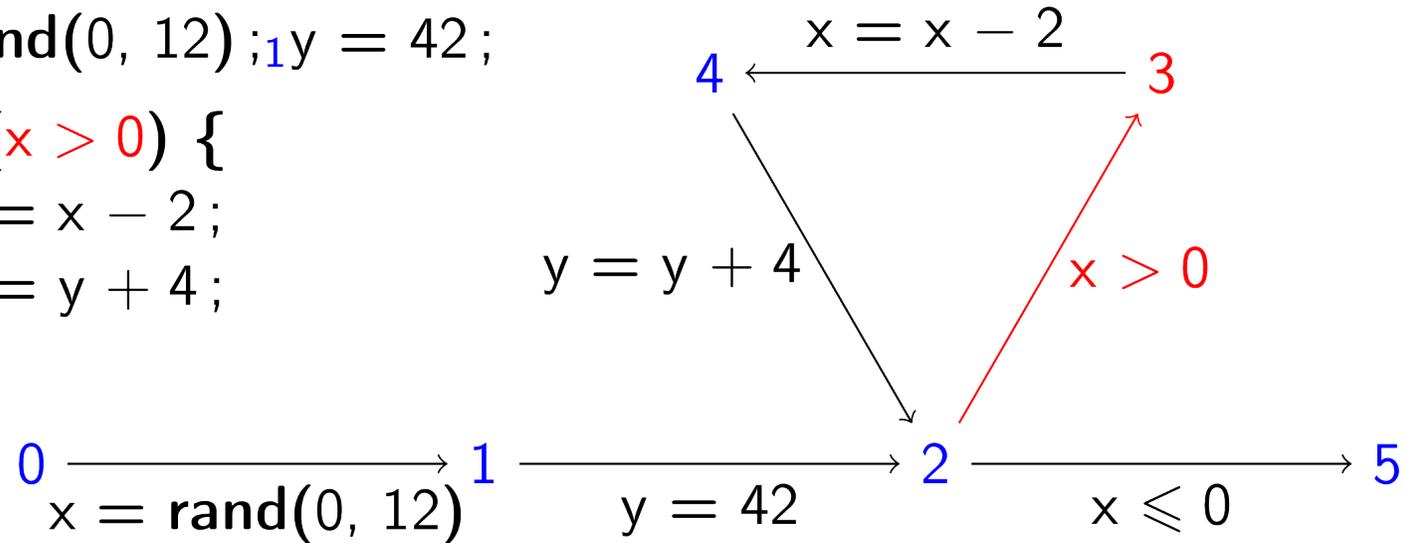
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)	(\top, \top)	(\top, \top)
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	$(\llbracket 0, 12 \rrbracket, \top)$
2	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \{42\})$	$(\llbracket -1, 12 \rrbracket, \llbracket 42, 46 \rrbracket)$
3	(\perp, \perp)	$(\llbracket 1, 12 \rrbracket, \{42\})$	$(\llbracket 1, 12 \rrbracket, \llbracket 42, 46 \rrbracket)$
4	(\perp, \perp)	$(\llbracket -1, 10 \rrbracket, \{42\})$	
5	(\perp, \perp)	$(\{0\}, \{42\})$	

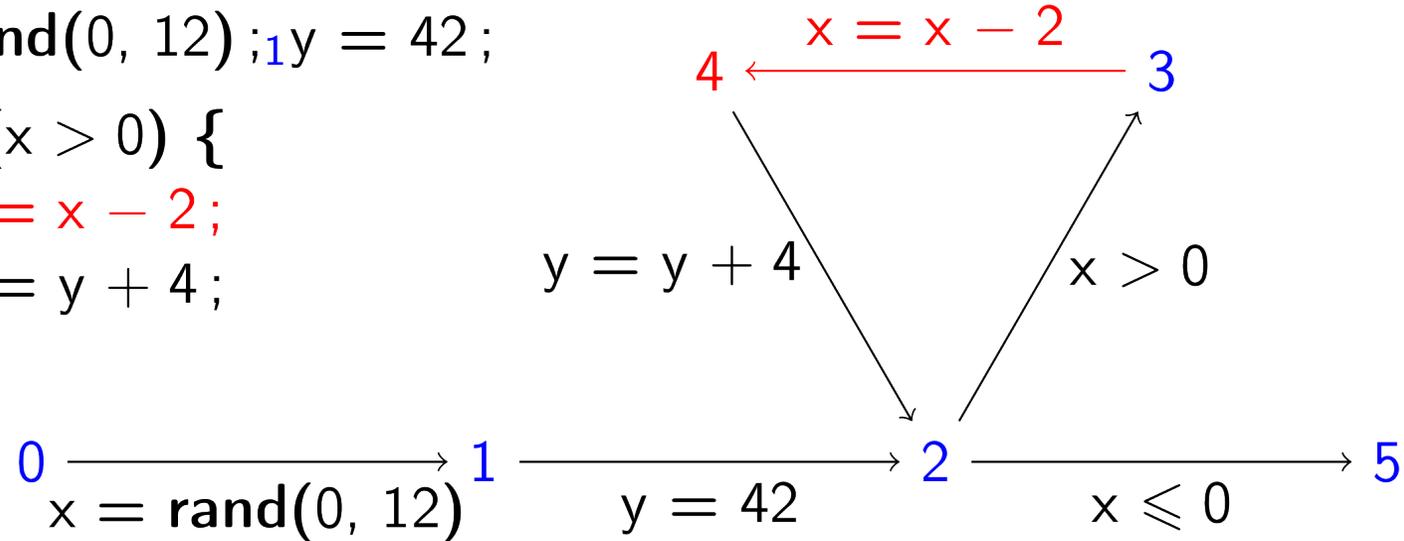
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)	(\top, \top)	(\top, \top)
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	$(\llbracket 0, 12 \rrbracket, \top)$
2	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \{42\})$	$(\llbracket -1, 12 \rrbracket, \llbracket 42, 46 \rrbracket)$
3	(\perp, \perp)	$(\llbracket 1, 12 \rrbracket, \{42\})$	$(\llbracket 1, 12 \rrbracket, \llbracket 42, 46 \rrbracket)$
4	(\perp, \perp)	$(\llbracket -1, 10 \rrbracket, \{42\})$	$(\llbracket -1, 10 \rrbracket, \llbracket 42, 46 \rrbracket)$
5	(\perp, \perp)	$(\{0\}, \{42\})$	

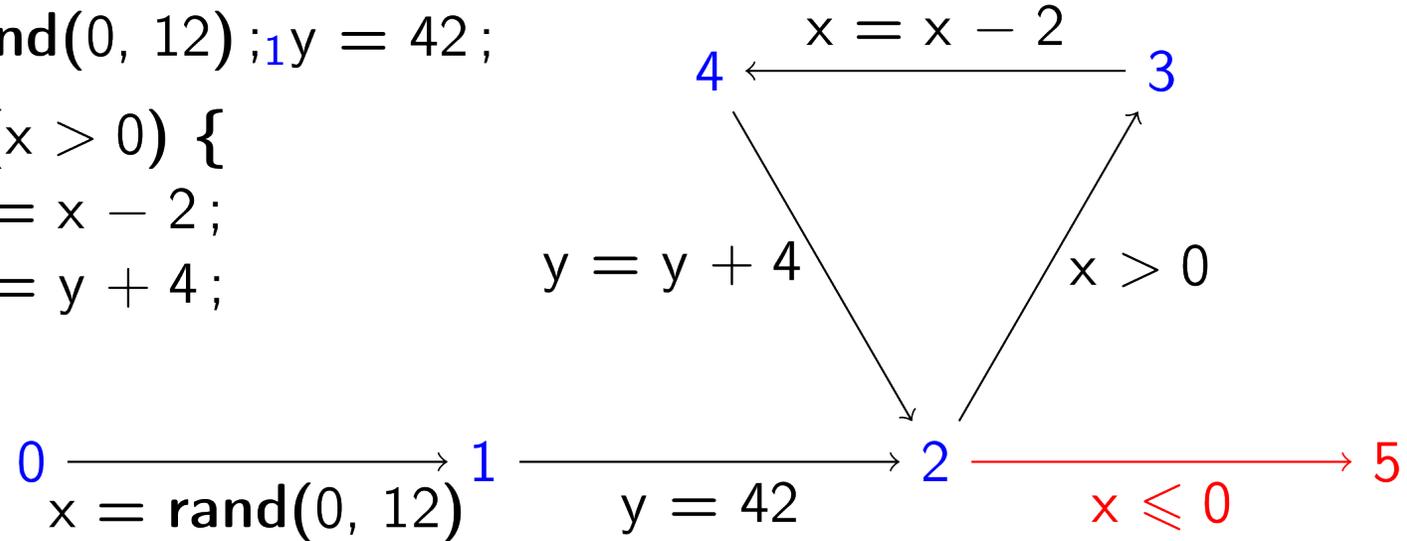
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)	(\top, \top)	(\top, \top)
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	$(\llbracket 0, 12 \rrbracket, \top)$
2	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \{42\})$	$(\llbracket -1, 12 \rrbracket, \llbracket 42, 46 \rrbracket)$
3	(\perp, \perp)	$(\llbracket 1, 12 \rrbracket, \{42\})$	$(\llbracket 1, 12 \rrbracket, \llbracket 42, 46 \rrbracket)$
4	(\perp, \perp)	$(\llbracket -1, 10 \rrbracket, \{42\})$	$(\llbracket -1, 10 \rrbracket, \llbracket 42, 46 \rrbracket)$
5	(\perp, \perp)	$(\{0\}, \{42\})$	$(\llbracket -1, 0 \rrbracket, \llbracket 42, 46 \rrbracket)$

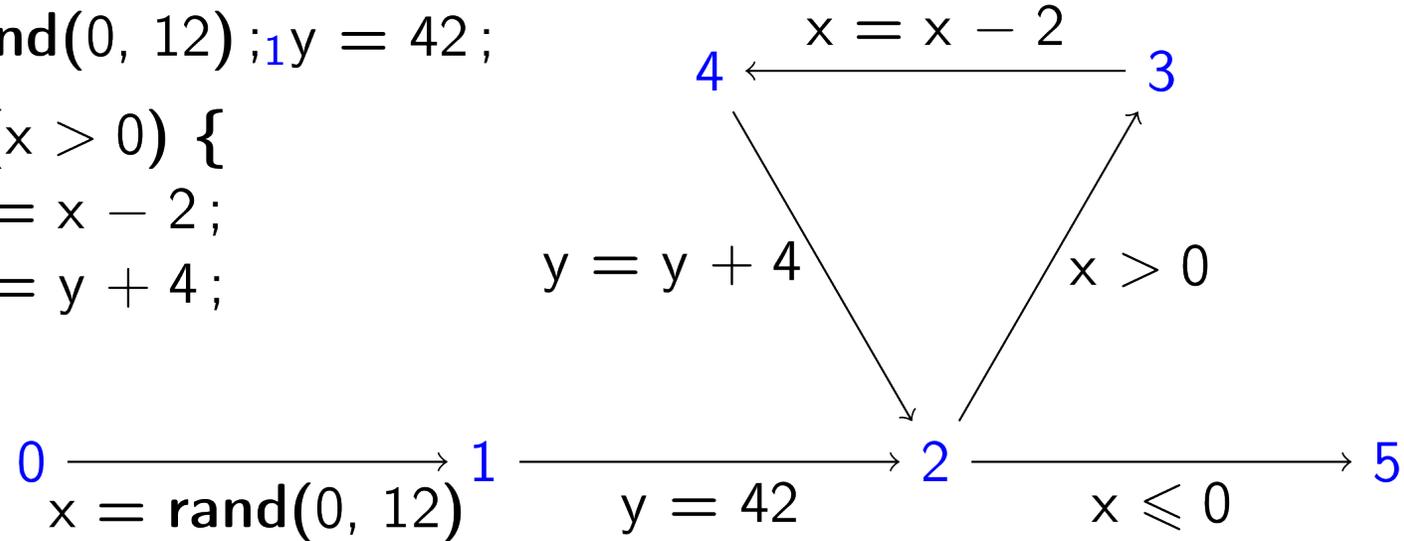
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$R_0^{\#i+1} = \top$$

$$R_1^{\#i+1} = R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket]$$

$$R_2^{\#i+1} = R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#}$$

$$R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket]$$

$$R_3^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket 1, +\infty \rrbracket]$$

$$R_4^{\#i+1} = R_3^{\#i+1} [x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket]$$

$$R_5^{\#i+1} = R_2^{\#i+1} [x \mapsto R_2^{\#i+1}(x)$$

$$\sqcap^{\#} \llbracket -\infty, 0 \rrbracket]$$

i	$R_i^{\#0}$	$R_i^{\#1}$	$R_i^{\#2}$
0	(\perp, \perp)	(\top, \top)	(\top, \top)
1	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \top)$	$(\llbracket 0, 12 \rrbracket, \top)$
2	(\perp, \perp)	$(\llbracket 0, 12 \rrbracket, \{42\})$	$(\llbracket -1, 12 \rrbracket, \llbracket 42, 46 \rrbracket)$
3	(\perp, \perp)	$(\llbracket 1, 12 \rrbracket, \{42\})$	$(\llbracket 1, 12 \rrbracket, \llbracket 42, 46 \rrbracket)$
4	(\perp, \perp)	$(\llbracket -1, 10 \rrbracket, \{42\})$	$(\llbracket -1, 10 \rrbracket, \llbracket 42, 46 \rrbracket)$
5	(\perp, \perp)	$(\{0\}, \{42\})$	$(\llbracket -1, 0 \rrbracket, \llbracket 42, 46 \rrbracket)$

The fixpoint is still far!

Correctness

The **abstract** semantic is an over-approximation of the **concrete** semantic.

For all $I \subseteq L$:

$$R_I \subseteq \gamma_{\text{nr}} \left(R_I^\# \right)$$

Termination

- **Problem:** This algorithm might not terminate!

Termination

- **Problem:** This algorithm might not terminate!
- The sequence $(F^n(\perp))_{n \in \mathbb{N}}$ might not converge

Termination

- **Problem:** This algorithm might not terminate!
- The sequence $(F^n(\perp))_{n \in \mathbb{N}}$ might not converge
- The sequence is increasing because F is monotonic

Termination

- **Problem:** This algorithm might not terminate!
- The sequence $(F^n(\perp))_{n \in \mathbb{N}}$ might not converge
- The sequence is increasing because F is monotonic
- If $D^\#$ is finite, then this would converge

Termination

- **Problem:** This algorithm might not terminate!
- The sequence $(F^n(\perp))_{n \in \mathbb{N}}$ might not converge
- The sequence is increasing because F is monotonic
- If $D^\#$ is finite, then this would converge
- But intervals have **infinitely increasing sequences:** $([0, n])_{n \in \mathbb{N}}$

Termination

- **Problem:** This algorithm might not terminate!
- The sequence $(F^n(\perp))_{n \in \mathbb{N}}$ might not converge
- The sequence is increasing because F is monotonic
- If $D^\#$ is finite, then this would converge
- But intervals have **infinitely increasing sequences:** $([0, n])_{n \in \mathbb{N}}$
- Even if it terminates, it could be slow..

Convergence Acceleration

- We introduce a new operator called *widening* ∇
- The *widening* is responsible for breaking **infinitely increasing sequences**
- It performs a jump forward

Widening

A **widening** ∇ is a binary operator $\nabla : \mathcal{D}^\# \times \mathcal{D}^\# \rightarrow \mathcal{D}^\#$ such that:

- $\forall x^\#, y^\#, \quad x^\# \sqcup^\# y^\# \sqsubseteq^\# x^\# \nabla y^\#$
- For all sequences $(x_n^\#)_{n \in \mathbb{N}}$, the following sequence **converges**:

$$\begin{cases} y_0^\# & = & x_0^\# \\ y_{i+1}^\# & = & y_i^\# \nabla x_{i+1}^\# \end{cases}$$

$$\begin{array}{c}
R^\# = F^{\#N}(\perp) = \text{lfp } F^\# \\
\uparrow \\
\vdots \\
\uparrow \\
R^{\#2} = F^\#(R^{\#1}) = F^{\#2}(\perp) \\
\uparrow \\
R^{\#1} = F^\#(R^{\#0}) = F^\#(\perp) \\
\uparrow \\
R^{\#0} = \perp
\end{array}$$

Without widening

$$\begin{array}{c}
 R^\# = F^{\#N}(\perp) = \text{lfp } F^\# \\
 \uparrow \\
 \vdots \\
 \uparrow \\
 R^{\#2} = F^\#(R^{\#1}) = F^{\#2}(\perp) \\
 \uparrow \\
 R^{\#1} = F^\#(R^{\#0}) = F^\#(\perp) \\
 \uparrow \\
 R^{\#0} = \perp
 \end{array}$$

Without widening

$$\begin{array}{c}
 R^\# = R^\# \nabla F^\#(R^\#) \\
 \left(\begin{array}{c} \text{lfp } F^\# \\ \vdots \end{array} \right) \\
 R^{\#2} = R^{\#1} \nabla F^\#(R^{\#1}) \\
 \left(\begin{array}{c} \\ \\ \end{array} \right) \\
 R^{\#1} = R^{\#0} \nabla F^\#(R^{\#0}) \\
 \left(\begin{array}{c} \\ \\ \end{array} \right) \\
 R^{\#0} = \perp
 \end{array}$$

With widening

Iterations with widening

- It is still a fixpoint: $R^\# = R^\# \nabla F^\#(R^\#)$ thus $F^\#(R^\#) \sqsubseteq^\# R^\#$

Iterations with widening

- It is still a fixpoint: $R^\# = R^\# \nabla F^\#(R^\#)$ thus $F^\#(R^\#) \sqsubseteq^\# R^\#$
- It is not the *least fixed point*

Iterations with widening

- It is still a fixpoint: $R^\# = R^\# \nabla F^\#(R^\#)$ thus $F^\#(R^\#) \sqsubseteq^\# R^\#$
- It is not the *least fixed point*
- It includes the *least fixed point*: $\text{lfp } F^\# \sqsubseteq^\# R^\#$

Interval widening

$$x^\# \nabla y^\# = \begin{cases} [a, b] & \text{si } x^\# = [a, b], y^\# = [c, d], c \geq a, d \leq b \\ [a, +\infty[& \text{si } x^\# = [a, b], y^\# = [c, d], c \geq a, d > b \\]-\infty, b] & \text{si } x^\# = [a, b], y^\# = [c, d], c < a, d \leq b \\]-\infty, +\infty[& \text{si } x^\# = [a, b], y^\# = [c, d], c < a, d > b \\ y^\# & \text{si } x^\# = \perp \\ x^\# & \text{si } y^\# = \perp \end{cases}$$

Interval widening

$$x^\# \nabla y^\# = \begin{cases} \llbracket a, b \rrbracket & \text{si } x^\# = \llbracket a, b \rrbracket, y^\# = \llbracket c, d \rrbracket, c \geq a, d \leq b \\ \llbracket a, +\infty[& \text{si } x^\# = \llbracket a, b \rrbracket, y^\# = \llbracket c, d \rrbracket, c \geq a, d > b \\ \llbracket -\infty, b \rrbracket & \text{si } x^\# = \llbracket a, b \rrbracket, y^\# = \llbracket c, d \rrbracket, c < a, d \leq b \\ \llbracket -\infty, +\infty[& \text{si } x^\# = \llbracket a, b \rrbracket, y^\# = \llbracket c, d \rrbracket, c < a, d > b \\ y^\# & \text{si } x^\# = \perp \\ x^\# & \text{si } y^\# = \perp \end{cases}$$

$$\llbracket 0, 1 \rrbracket \nabla \llbracket 0, 2 \rrbracket = \llbracket 0, +\infty[$$

$$\llbracket 0, 2 \rrbracket \nabla \llbracket 0, 1 \rrbracket = \llbracket 0, 2 \rrbracket$$

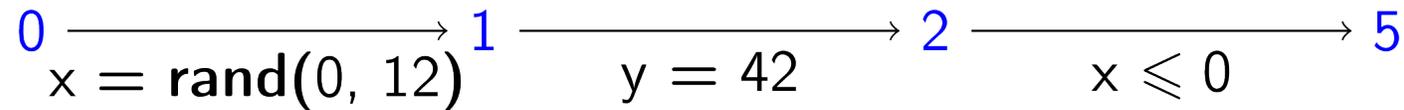
0 $x = \text{rand}(0, 12)$; 1 $y = 42$;

while 2 $(x > 0)$ {

3 $x = x - 2$;

4 $y = y + 4$;

} 5



$$\begin{aligned}
 R_0^{\#i+1} &= \top \\
 R_1^{\#i+1} &= R_0^{\#i+1} [x \mapsto \llbracket 0, 12 \rrbracket] \\
 R_2^{\#i+1} &= R_1^{\#i+1} [y \mapsto \llbracket 42, 42 \rrbracket] \sqcup_{\text{nr}}^{\#} \\
 &\quad R_4^{\#i} [y \mapsto R_4^{\#i}(y) +^{\#} \llbracket 4, 4 \rrbracket] \\
 R_3^{\#i+1} &= R_2^{\#i+1} \left[x \mapsto R_2^{\#i+1}(x) \right. \\
 &\quad \left. \sqcap^{\#} \llbracket 1, +\infty \rrbracket \right] \\
 R_4^{\#i+1} &= R_3^{\#i+1} \left[x \mapsto R_3^{\#i+1}(x) -^{\#} \llbracket 2, 2 \rrbracket \right] \\
 R_5^{\#i+1} &= R_2^{\#i+1} \left[x \mapsto R_2^{\#i+1}(x) \right. \\
 &\quad \left. \sqcap^{\#} \llbracket -\infty, 0 \rrbracket \right]
 \end{aligned}$$

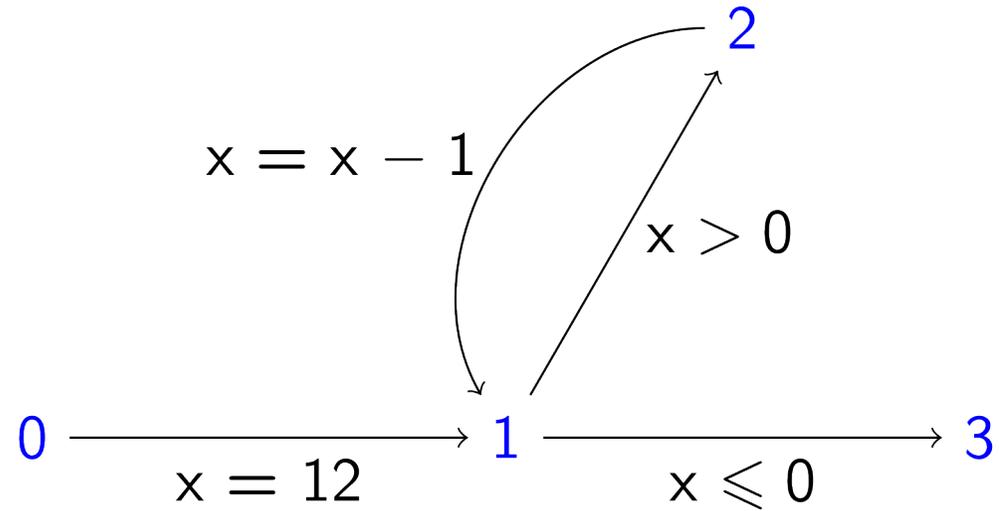
$$\begin{aligned}
 R_0^{\#} &= \top_{\text{nr}} \\
 R_1^{\#} &= (\llbracket 0, 12 \rrbracket, \top) \\
 R_2^{\#} &= (\llbracket -\infty, 12 \rrbracket, \llbracket 42, +\infty \rrbracket) \\
 R_3^{\#} &= (\llbracket 1, 12 \rrbracket, \llbracket 42, +\infty \rrbracket) \\
 R_4^{\#} &= (\llbracket -1, 10 \rrbracket, \llbracket 46, +\infty \rrbracket) \\
 R_5^{\#} &= (\llbracket -\infty, 0 \rrbracket, \llbracket 42, +\infty \rrbracket)
 \end{aligned}$$

Widening

- Widening allows the algorithm to terminate quickly
- But it might cause a loss of precision
- In practice, we only use the widening after a few iterations
- We can also use a widening with a threshold

$0x = 12;$

while $1(x > 0)$ {
 $2x = x - 1;$
} 3



$$\begin{aligned}
 R_0^{\#i+1} &= \top \\
 R_1^{\#i+1} &= R_1^{\#i} \nabla_{\text{nr}} \left(R_0^{\#i+1} [x \mapsto \llbracket 12, 12 \rrbracket] \sqcup_{\text{nr}}^{\#} \right. \\
 &\quad \left. R_2^{\#i} [y \mapsto R_2^{\#i}(x) -^{\#} \llbracket 1, 1 \rrbracket] \right) \\
 R_3^{\#i+1} &= R_1^{\#i+1} \left[x \mapsto R_1^{\#i+1}(x) \right. \\
 &\quad \left. \sqcap^{\#} \llbracket -\infty, 0 \rrbracket \right]
 \end{aligned}$$

l	$R_l^{\#0}$	$R_l^{\#1}$	$R_l^{\#2}$	$R_l^{\#3}$
0	\perp	\perp	\perp	\perp
1	\perp	$\llbracket 12, 12 \rrbracket$	$\llbracket -\infty, 12 \rrbracket$	$\llbracket -\infty, 12 \rrbracket$
2	\perp	$\llbracket 12, 12 \rrbracket$	$\llbracket 1, 12 \rrbracket$	$\llbracket 1, 12 \rrbracket$
3	\perp	\perp	$\llbracket -\infty, 0 \rrbracket$	$\llbracket -\infty, 0 \rrbracket$

Regain precision

- We introduce a new operator called narrowing Δ
- Perform decreasing iterations to regain precision

Narrowing

A **narrowing** Δ is a binary operator $\Delta: \mathcal{D}^\# \times \mathcal{D}^\# \rightarrow \mathcal{D}^\#$ such that:

- $\forall x^\#, y^\#, \quad x^\# \sqcap^\# y^\# \sqsubseteq^\# x^\# \Delta y^\# \sqsubseteq^\# x^\#$
- For all sequences $(x_n^\#)_{n \in \mathbb{N}}$, the following sequence **converges**:

$$\begin{cases} y_0^\# & = & x_0^\# \\ y_{i+1}^\# & = & y_i^\# \Delta x_{i+1}^\# \end{cases}$$

$$R^\# = R^\# \nabla F^\#(R^\#)$$

$$R^{\#1} = R^\# \Delta F^\#(R^\#)$$

\vdots

$$R^{\#'} = R^{\#'} \Delta F^\#(R^{\#'})$$

lfp $F^\#$

\vdots

$$R^{\#0} = \perp$$

$$R^\# = R^\# \nabla F^\#(R^\#)$$

$$R^{\#1} = R^\# \Delta F^\#(R^\#)$$

\vdots

$$R^{\#'} = R^{\#'} \Delta F^\#(R^{\#'})$$

$\text{lfp } F^\#$

\vdots

$$R^{\#0} = \perp$$

$$\text{lfp } F^\# \sqsubseteq^\# R^{\#'}$$

Interval narrowing

$$x^\# \Delta y^\# = \begin{cases} \llbracket a, d \rrbracket & \text{si } x^\# = \llbracket a, +\infty \llbracket, y^\# = \llbracket c, d \rrbracket \\ \llbracket c, b \rrbracket & \text{si } x^\# = \llbracket -\infty, b \rrbracket, y^\# = \llbracket c, d \rrbracket \\ \llbracket c, d \rrbracket & \text{si } x^\# = \llbracket -\infty, +\infty \llbracket, y^\# = \llbracket c, d \rrbracket \\ x^\# & \text{sinon} \end{cases}$$

Interval narrowing

$$x^\# \Delta y^\# = \begin{cases} \llbracket a, d \rrbracket & \text{si } x^\# = \llbracket a, +\infty \llbracket, y^\# = \llbracket c, d \rrbracket \\ \llbracket c, b \rrbracket & \text{si } x^\# = \llbracket -\infty, b \rrbracket, y^\# = \llbracket c, d \rrbracket \\ \llbracket c, d \rrbracket & \text{si } x^\# = \llbracket -\infty, +\infty \llbracket, y^\# = \llbracket c, d \rrbracket \\ x^\# & \text{sinon} \end{cases}$$

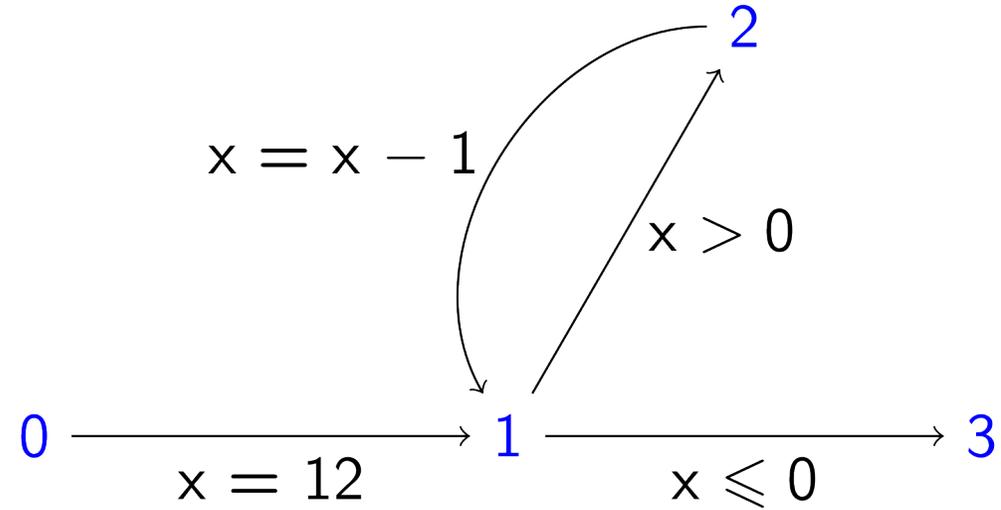
$$\llbracket 0, +\infty \llbracket \Delta \llbracket 0, 1 \rrbracket = \llbracket 0, 1 \rrbracket$$

$$\llbracket 0, 2 \rrbracket \Delta \llbracket 0, 1 \rrbracket = \llbracket 0, 2 \rrbracket$$

```

0 x = 12;
while 1 (x > 0) {
    2 x = x - 1;
}3

```



$$\begin{aligned}
 R_0^{\#i+1} &= \top \\
 R_1^{\#i+1} &= R_1^{\#i} \nabla_{\text{nr}} \left(R_0^{\#i+1} [x \mapsto \llbracket 12, 12 \rrbracket] \sqcup_{\text{nr}} \right. \\
 &\quad \left. R_2^{\#i} [y \mapsto R_2^{\#i}(x) -^{\#} \llbracket 1, 1 \rrbracket] \right) \\
 R_3^{\#i+1} &= R_1^{\#i+1} [x \mapsto R_1^{\#i+1}(x) \\
 &\quad \sqcap^{\#} \llbracket -\infty, 0 \rrbracket]
 \end{aligned}$$

$$\begin{aligned}
 R_0^{\#} &= \top_{\text{nr}} \\
 R_1^{\#} &= \llbracket 0, 12 \rrbracket \\
 R_2^{\#} &= \llbracket 1, 12 \rrbracket \\
 R_3^{\#} &= \llbracket 0, 0 \rrbracket
 \end{aligned}$$

Abstract Domains

Domain	Constraints	Complexity
Interval	$x \in [a, b]$	n
Congruence	$x \in aZ+b$	n
Gauge	$x \in [a*i + b*k + \dots, a'*i + b'*k + \dots]$	$K*n$
Difference Bound Matrices	$x - y \in [a, b]$	n^3
Octagon	$x \pm y \in [a, b]$	n^3
Polyhedra	$a*x + b*y + \dots + c \leq 0$	Exponential

Thank you.

Questions?

maxime.arthaud@nasa.gov