

Program Analysis

5. Abstract Interpretation (1): Concrete Semantics

Kihong Heo

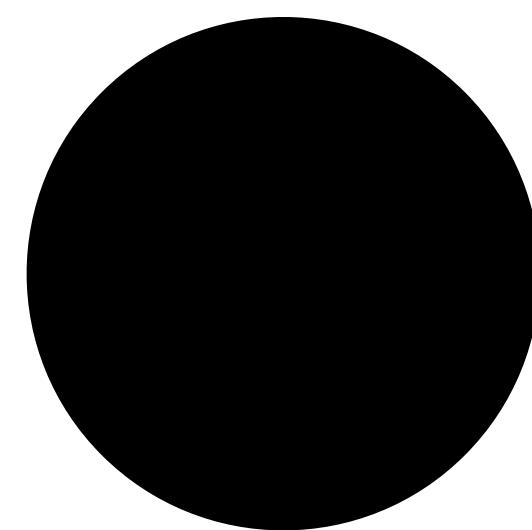


Abstract Interpretation (요약 해석)

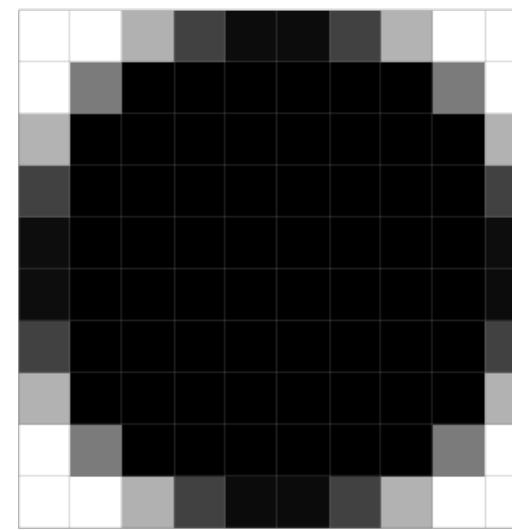
- A **powerful framework** for designing **correct** static analysis
 - Framework: given some inputs, a static analysis comes out
 - Powerful: all static analyses are understood in this framework (e.g., type systems, data-flow analysis, etc)
 - Correct: mathematically proven
- Established by Patrick and Radhia Cousot
 - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*, 1977
 - *Systematic Design of Program Analysis Frameworks*, 1979

Abstract?

- Concrete (execution, dynamic) vs Abstract (analysis, static)
- Without abstraction, it is undecidable to subsume all possible behavior of SW
 - Recall the Rice's theorem and the Halting problem

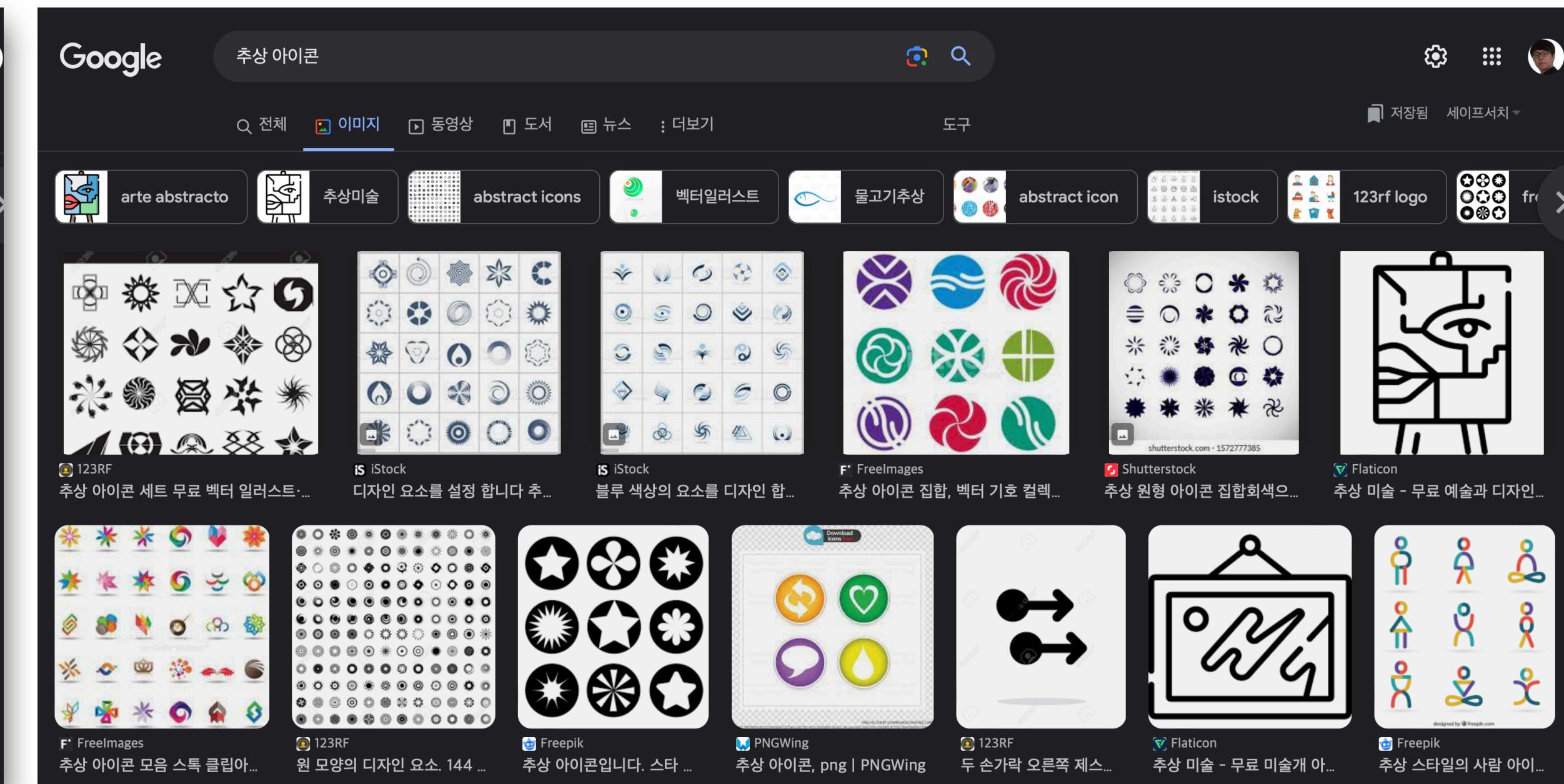
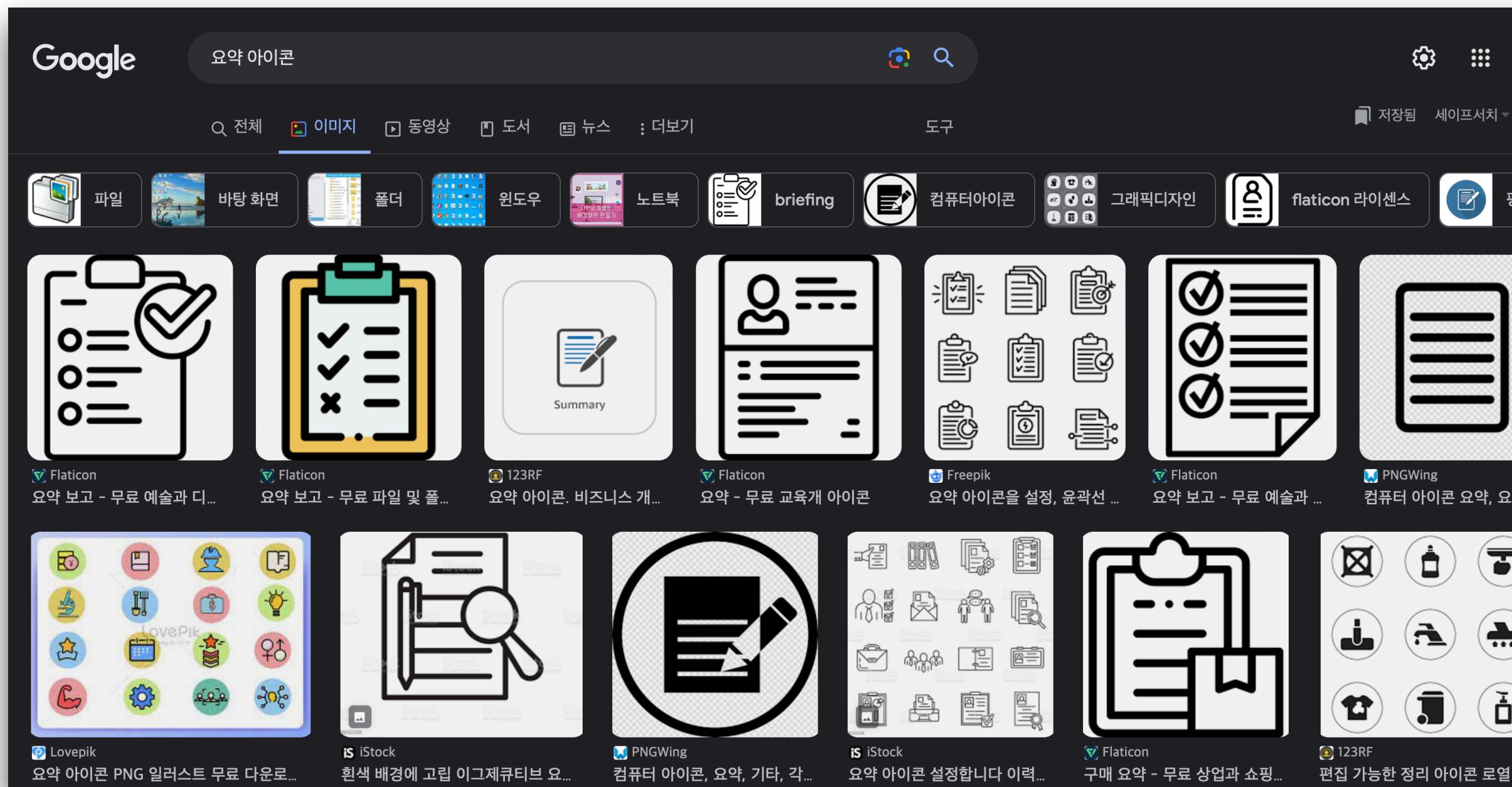


Concrete



Abstract

요약 vs 추상



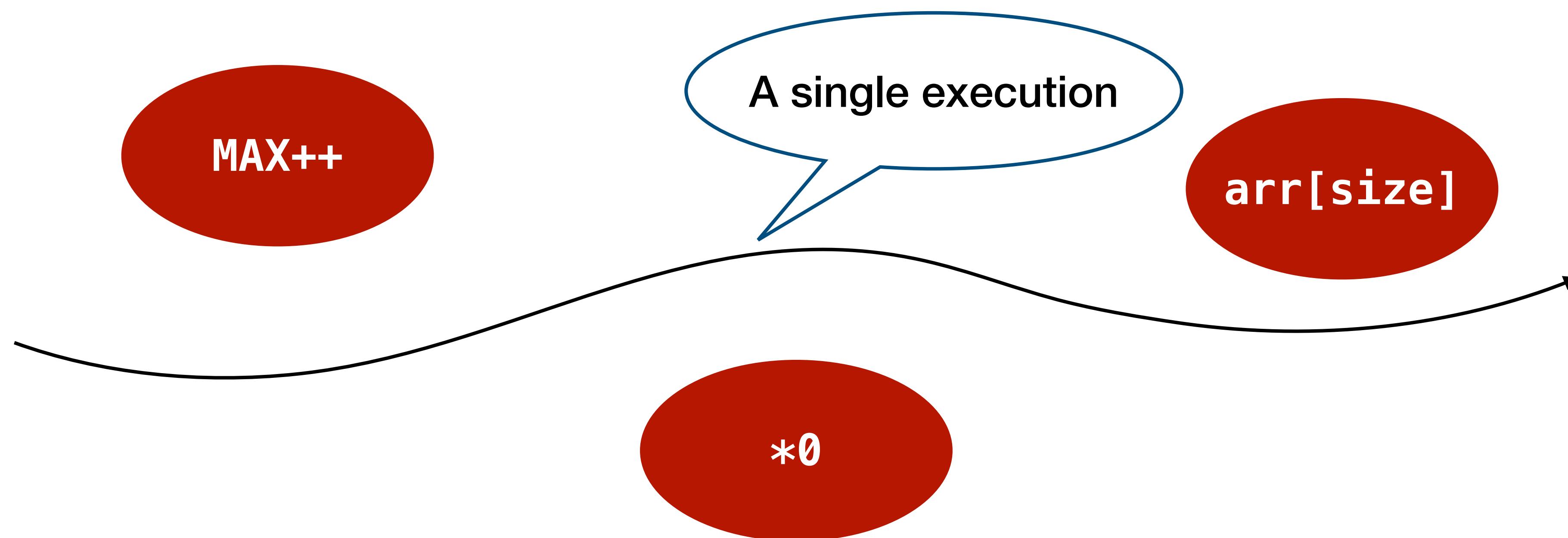
Example

```
x = 3;  
while (*) {  
    x += 2;  
}  
x -= 1;  
print(x);
```

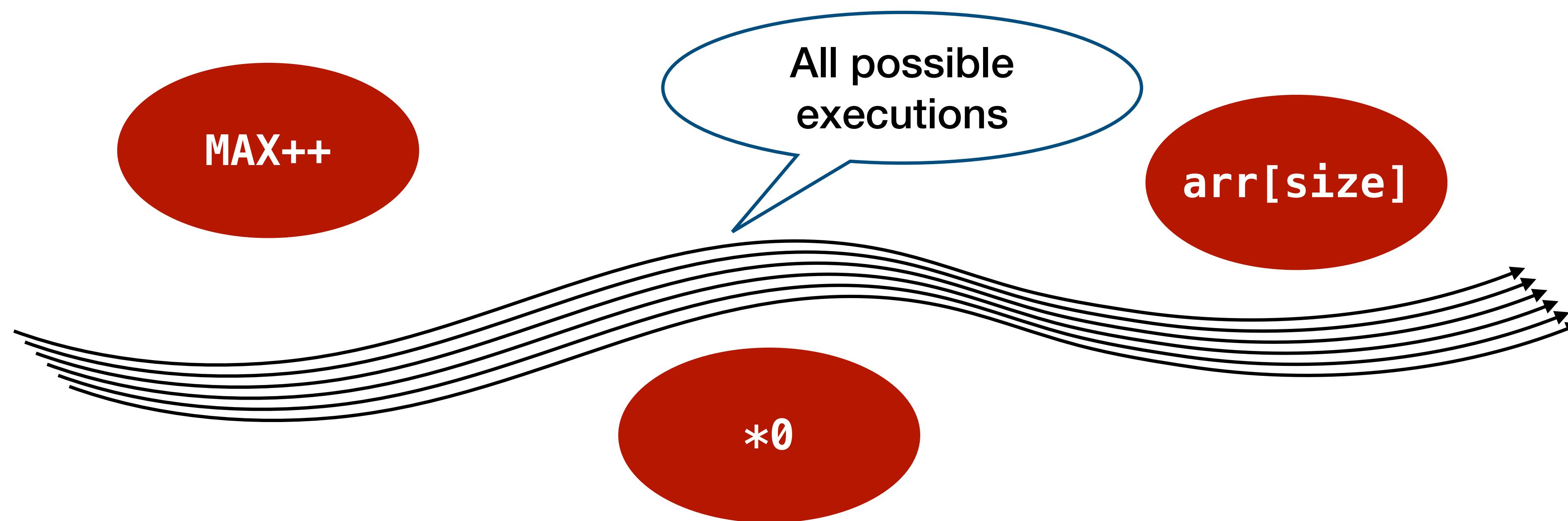
Q: What are the possible output values?

- Concrete interpretation : 2, 4, ..., uncomputable (infinitely many possibilities)
- Abstract interpretation 1 : “integers” (good)
- Abstract interpretation 2 : “positive integers” (better)
- Abstract interpretation 3 : “positive even integers” (best)

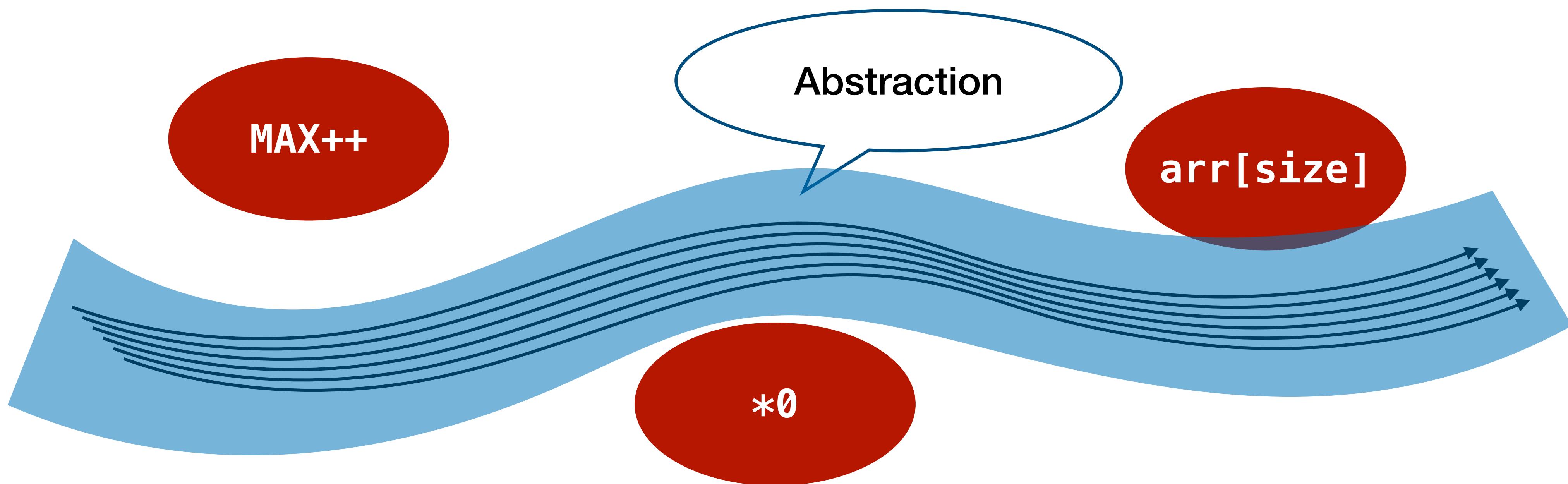
Abstraction of Executions



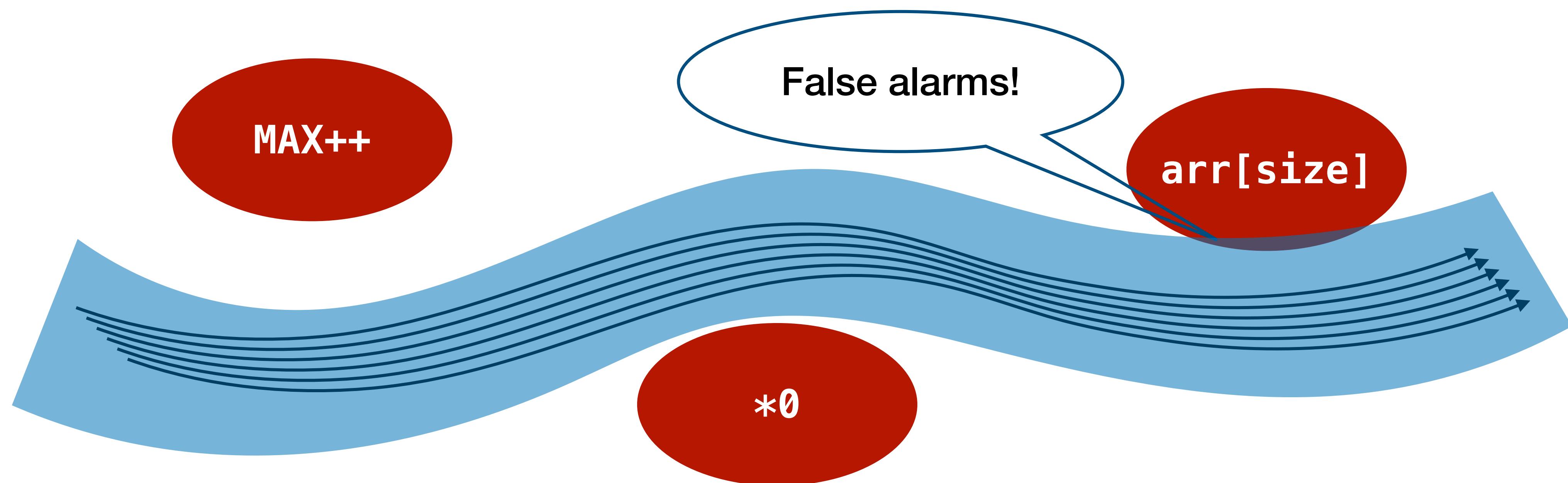
Abstraction of Executions



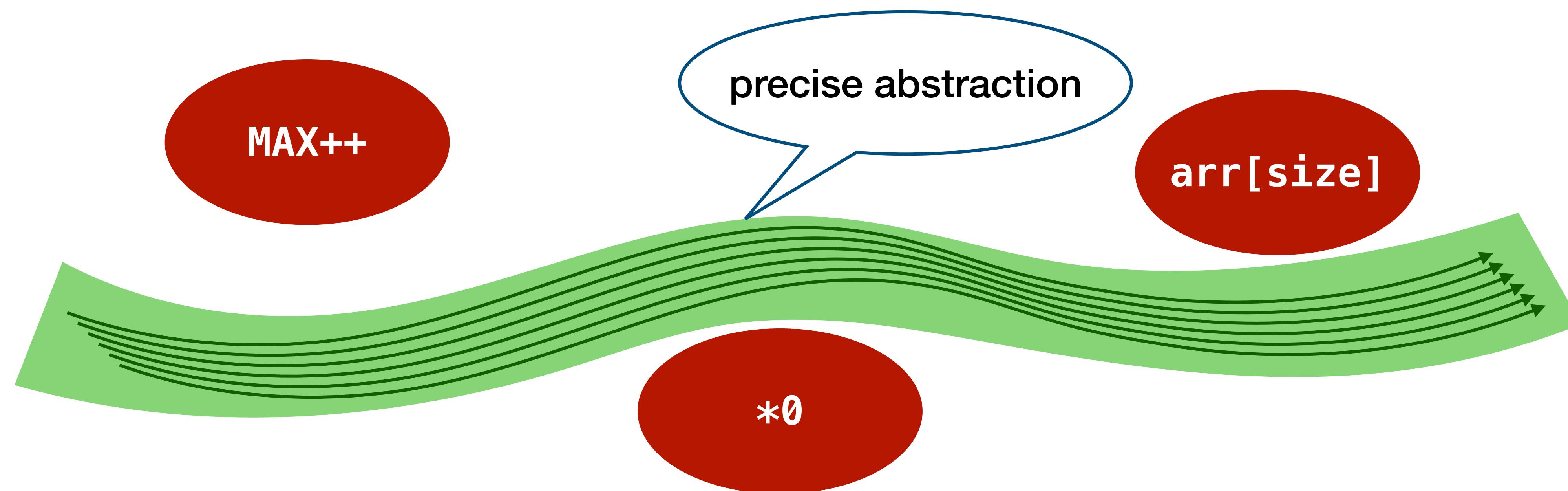
Abstraction of Executions



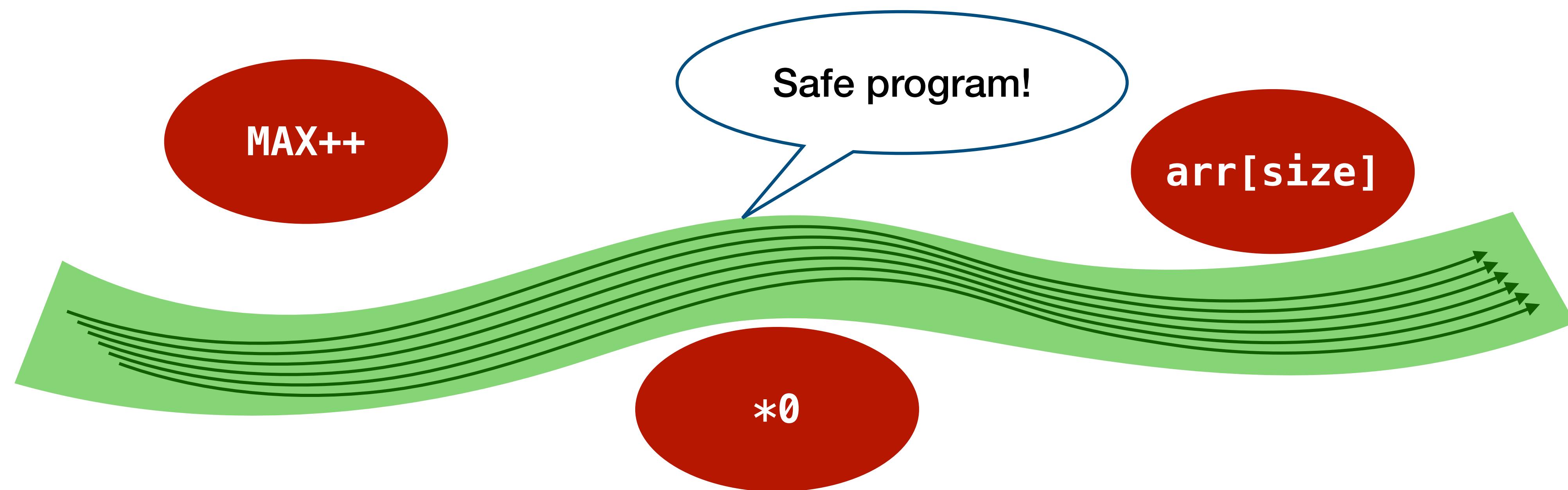
Abstraction of Executions



Abstraction of Executions



Abstraction of Executions



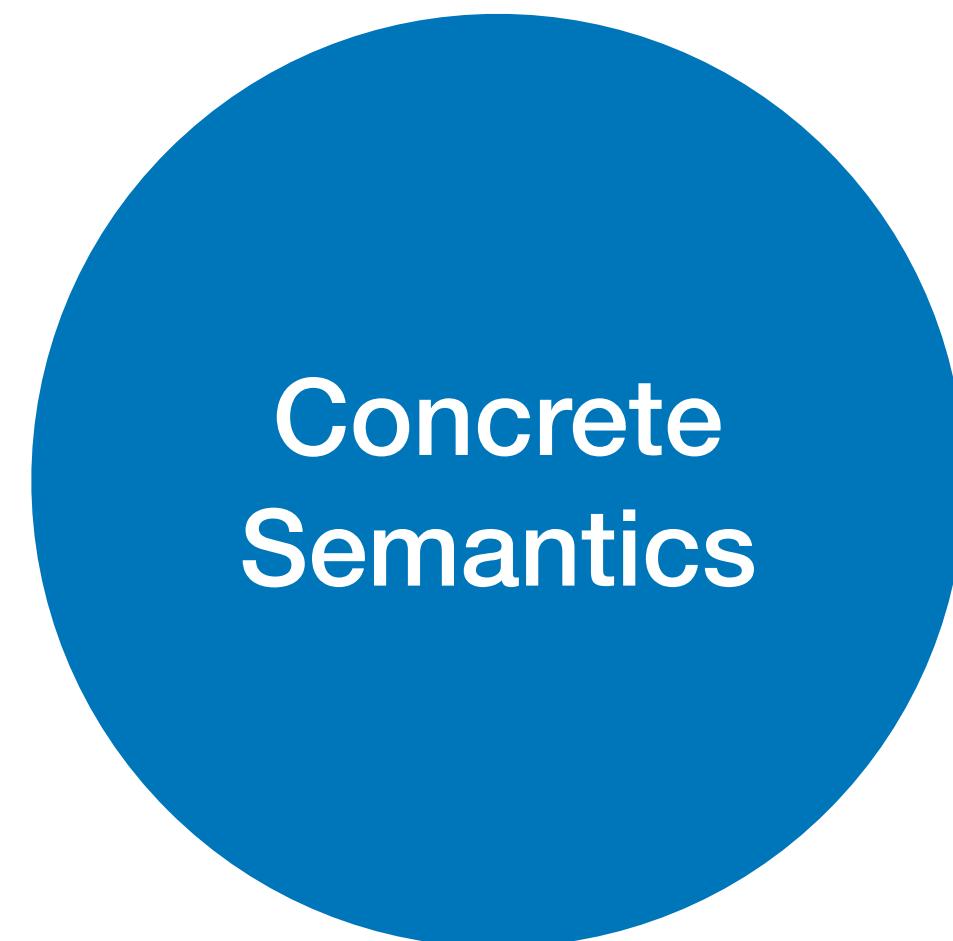
How to analyze?

- Interpret the target program
 - with abstract semantics (= analyzer's concern)
 - not concrete semantics (= interpreter's and compiler's concern)

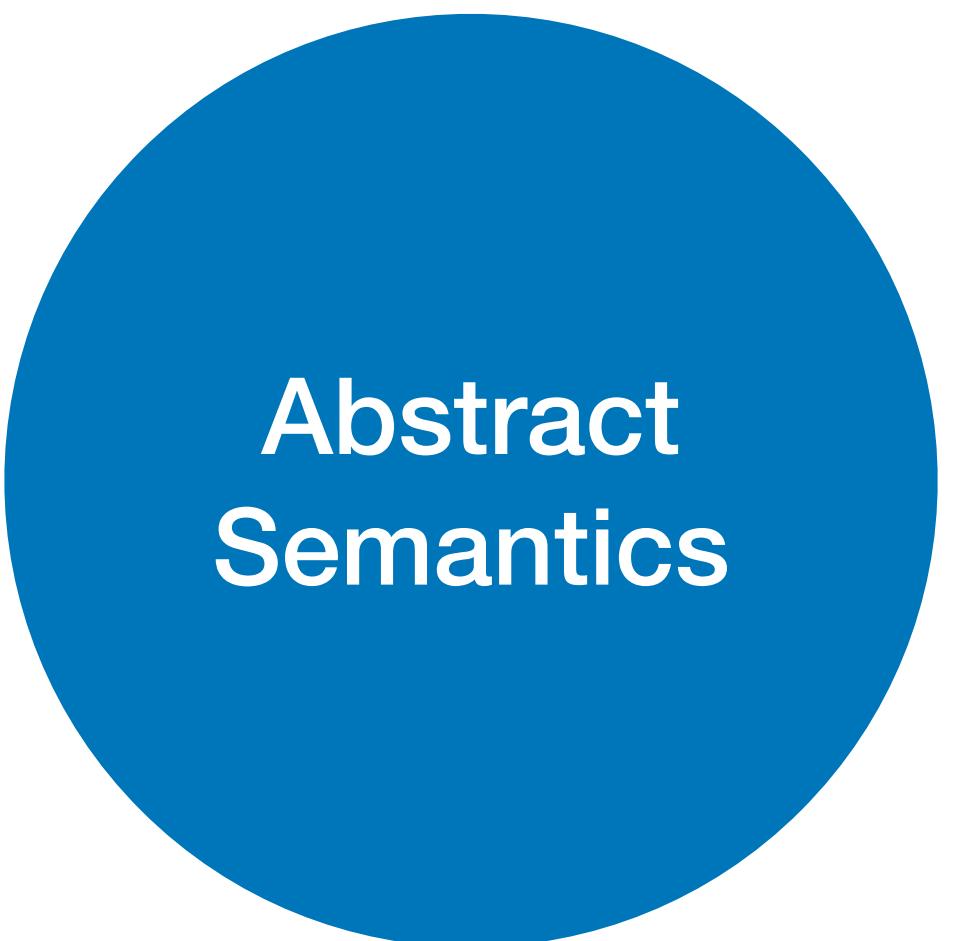
- Example

	Concrete	Abstract 1	Abstract 2	Abstract 3
x = 3;	{3}	Int	Pos	PosOdd
while (*) {				
x += 2;				
}	{3, 5, 7, ...}	Int	Pos	PosOdd
x -= 1;	{2, 4, 6, ...}	Int	Pos	PosEven
print(x);				

Principles



?
≈



- How to guarantee soundness?
- How to guarantee termination?
- How to design more precise abstraction?
- How to compute abstract semantics?

Practice



- Guidance for a lot of design choices in practice such as
 - Soundness vs Scalability vs Precision vs Usability vs ...
 - Characteristics of target programs and properties
 - Optimizations of program analyzers

Abstract Interpretation Framework

- Abstract interpretation concerns
 - Concrete semantics: $\llbracket C \rrbracket = \text{Ifp } F \in \mathbb{D}$
 - Abstract semantics: $\llbracket C \rrbracket^\sharp = \bigsqcup_{i \geq 0} F^{\sharp i}(\perp) \in \mathbb{D}^\sharp$
- Requirements:
 - Relationship between \mathbb{D} and \mathbb{D}^\sharp
 - Relationship between $F \in \mathbb{D} \rightarrow \mathbb{D}$ and $F^\sharp \in \mathbb{D}^\sharp \rightarrow \mathbb{D}^\sharp$
- Guarantees:
 - Correctness (soundness): $\llbracket C \rrbracket \approx \llbracket C \rrbracket^\sharp$
 - Computability: $\llbracket C \rrbracket^\sharp$ is computable within finite time

Design of Static Analysis

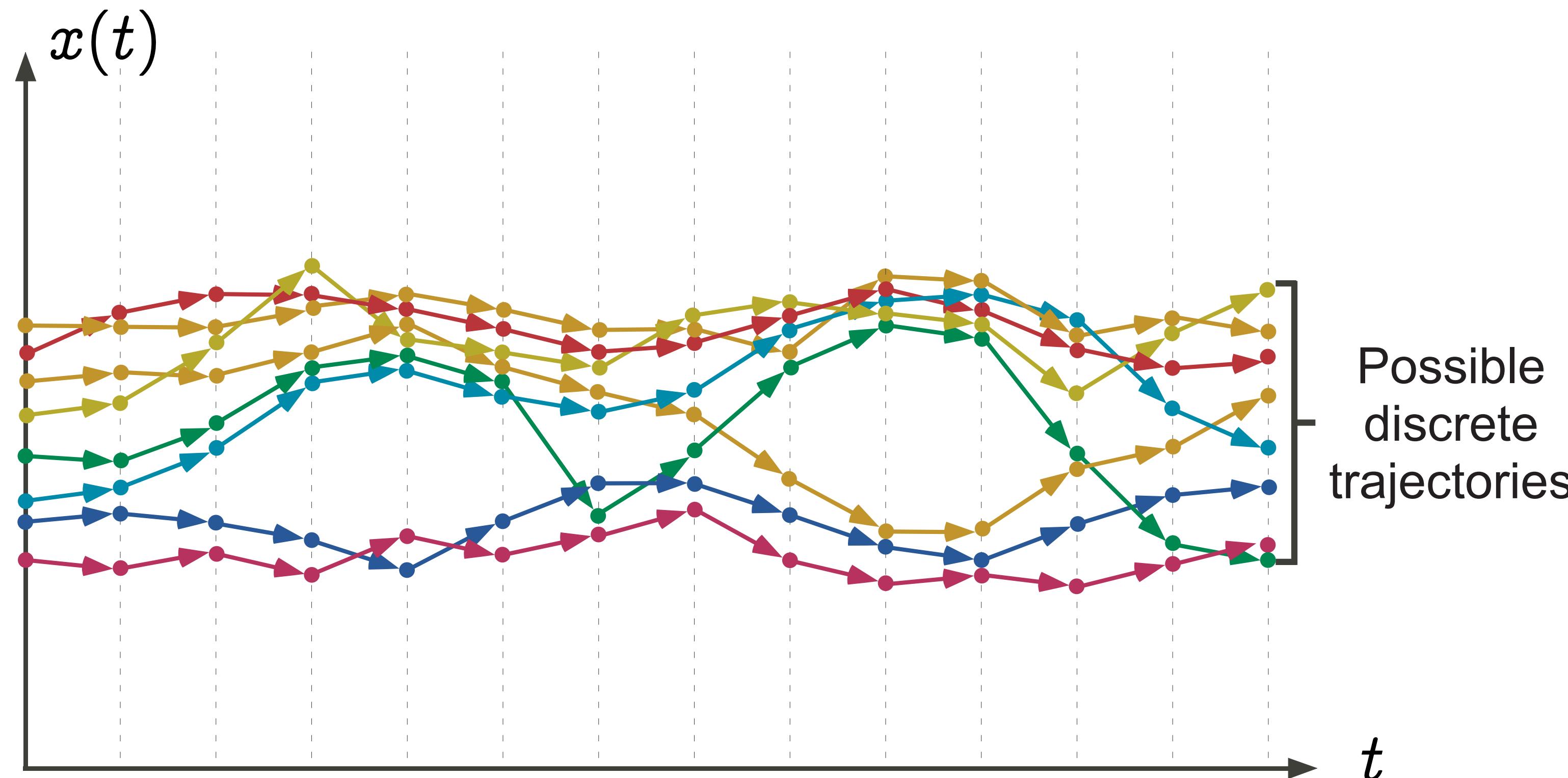
- Goal: **conservative** and **terminating** static analysis
- Design principles:
 - Define **concrete semantics**
 - Define **abstract semantics** (sound w.r.t the concrete semantics)
- Computation & implementation:
 - Abstract semantics of a program: **the least fixed point** of the semantic function
 - Static analyzer: **compute** the least fixed point within **finite time**

Define Standard Semantics

- Formalization of a **single program execution**
 - Recall Lecture 2 and 3 (operational and denotation semantics)
- What to describe: different choices depending on the purpose
 - E.g., denotational, operational, etc
- In this lecture, we will use denotational semantics
 - Recall the denotational semantics the simple imperative language

$$[\![C]\!] : \mathbb{M} \rightarrow \mathbb{M}$$

Define Standard Semantics



*from Patrick Cousot's slides

Standard Semantics

- Define a semantic domain $\mathbb{D} = \mathbb{M} \rightarrow \mathbb{M}$ (CPO)
- Define a semantic function $F : \mathbb{D} \rightarrow \mathbb{D}$ (continuous)
- Semantics of a program: the least fixed point of F

$$\text{lfp } F = \bigsqcup_{i \geq 0} F^i(\perp)$$

Standard Semantics of Commands

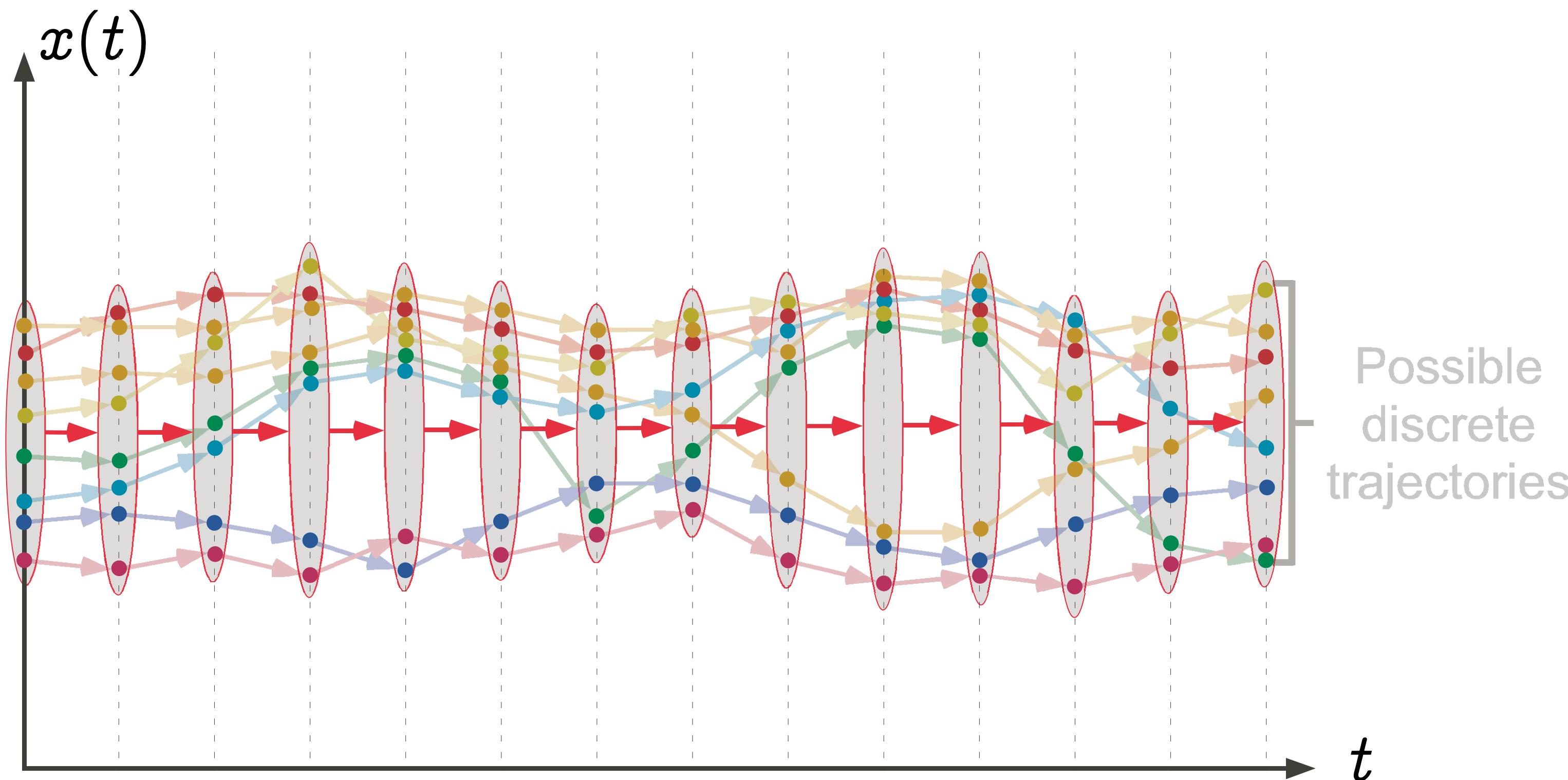
$$\begin{aligned}\llbracket C \rrbracket &: \mathbb{M} \rightarrow \mathbb{M} \\ \llbracket \text{skip} \rrbracket &= \lambda m. m \\ \llbracket C_0 ; C_1 \rrbracket &= \lambda m. \llbracket C_1 \rrbracket(\llbracket C_0 \rrbracket(m)) \\ \llbracket x := E \rrbracket &= \lambda m. m\{x \mapsto \llbracket E \rrbracket(m)\} \\ \llbracket \text{input}(x) \rrbracket &= \lambda m. m\{x \mapsto n\} \\ \llbracket \text{if } B \text{ then } C_1 \text{ else } C_2 \rrbracket &= \lambda m. \begin{cases} \llbracket C_1 \rrbracket(m) & \text{if } \llbracket B \rrbracket(m) = \text{true} \\ \llbracket C_2 \rrbracket(m) & \text{if } \llbracket B \rrbracket(m) = \text{false} \end{cases} \\ \llbracket \text{while } B \text{ } C \rrbracket &= \text{lfp} \lambda X. \left(\lambda m. \begin{cases} X(\llbracket C \rrbracket(m)) & \text{if } \llbracket B \rrbracket(m) = \text{true} \\ m & \text{if } \llbracket B \rrbracket(m) = \text{false} \end{cases} \right)\end{aligned}$$

Define Concrete Semantics

- Formalization of **all possible** program executions
 - So-called collecting semantics
 - Usually a simple extension of the standard semantics
- What to describe: different choices depending on the purposes (recall, property)
 - Some are more expressive than others
 - E.g., traces (sequence of states), reachable states (set of states), etc
- In this lecture, we will use reachable states for concrete semantics

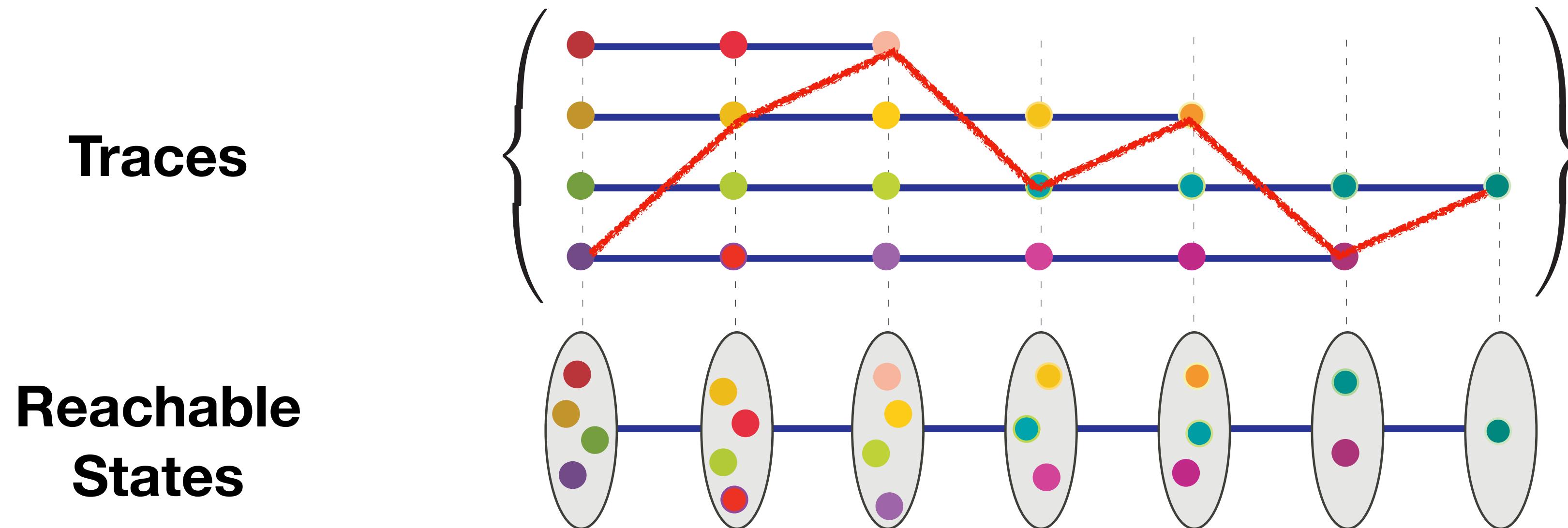
$$\llbracket C \rrbracket : \mathbb{M} \rightarrow \mathbb{M} \xrightarrow{\text{collecting}} \llbracket C \rrbracket_{\wp} : \wp(\mathbb{M}) \rightarrow \wp(\mathbb{M})$$

Transitions of Sets of States



*from Patrick Cousot's slides

Traces vs Reachable States



Can Answer:

- Can variable p be NULL at line 10?
- Can buffer index i be larger than size s?
- ...

Can't Answer:

- Is the value of p at line 5 the same as that of at line 1?
- Does the red trace exist?
- ...

*from Patrick Cousot's slides

Concrete Semantics

- Define a concrete domain \mathbb{D} (CPO)
- Define a semantic function $F : \mathbb{D} \rightarrow \mathbb{D}$ (continuous)
- Then the concrete semantics is defined as the least fixed point of the semantic function $F :$

$$\text{lfp } F = \bigsqcup_{i \geq 0} F^i(\perp)$$

Example

- Define a concrete semantics of the simple language using **denotational semantics**
 - Concrete domain $\mathbb{D} = \wp(\mathbb{M}) \rightarrow \wp(\mathbb{M})$
 - Define a semantic function $F : \mathbb{D} \rightarrow \mathbb{D}$
 - Concrete semantics $\text{lfp } F \in \mathbb{D}$
- Q: How to define F ?

Concrete Semantics of Expressions

$$\llbracket E \rrbracket_{\wp} : \wp(\mathbb{M}) \rightarrow \wp(\mathbb{Z})$$

$$\llbracket n \rrbracket_{\wp} = \lambda M. \{n\}$$

$$\llbracket x \rrbracket_{\wp} = \lambda M. \{m(x) \mid m \in M\}$$

$$\llbracket E_1 \odot E_2 \rrbracket_{\wp} = \lambda M. \{\llbracket E_1 \rrbracket(m) \odot \llbracket E_2 \rrbracket(m) \mid m \in M\}$$

$$\llbracket B \rrbracket_{\wp} : \wp(\mathbb{M}) \rightarrow \wp(\mathbb{M})$$

$$\llbracket \text{true} \rrbracket_{\wp} = \lambda M. M$$

$$\llbracket \text{false} \rrbracket_{\wp} = \lambda M. \emptyset$$

$$\llbracket E_1 \oslash E_2 \rrbracket_{\wp} = \lambda M. \{m \in M \mid \llbracket E_1 \rrbracket(m) \oslash \llbracket E_2 \rrbracket(m) = \text{true}\}$$

Concrete Semantics of Commands

$$\llbracket C \rrbracket_{\wp} : \wp(\mathbb{M}) \rightarrow \wp(\mathbb{M})$$

$$\llbracket \text{skip} \rrbracket_{\wp} = \lambda M. M$$

$$\llbracket C_0 ; C_1 \rrbracket_{\wp} = \lambda M. \llbracket C_1 \rrbracket_{\wp} \circ \llbracket C_0 \rrbracket_{\wp}(M)$$

$$\llbracket x := E \rrbracket_{\wp} = \lambda M. \{m \{x \mapsto v\} \mid v \in \llbracket E \rrbracket_{\wp}(m)\}$$

$$\llbracket \text{input}(x) \rrbracket_{\wp} = \lambda M. \{m \{x \mapsto n\} \mid m \in M, n \in \mathbb{Z}\}$$

$$\llbracket \text{if } B \text{ then } C_1 \text{ else } C_2 \rrbracket_{\wp} = \lambda M. \llbracket C_1 \rrbracket_{\wp} \circ \llbracket B \rrbracket_{\wp}(M) \cup \llbracket C_2 \rrbracket_{\wp} \circ \llbracket \neg B \rrbracket_{\wp}(M)$$

$$\llbracket \text{while } B \text{ } C \rrbracket_{\wp} = \lambda M. \llbracket \neg B \rrbracket_{\wp} (\text{lfp} \lambda X. M \cup \llbracket C \rrbracket_{\wp} \circ \llbracket B \rrbracket_{\wp}(X))$$

Summary

- Abstract interpretation: a **framework** for designing correct static analysis
- Concrete semantics: collection of **all possible behaviors** of a program
 - Usually a simple extension of the standard semantics
 - Defined as a least fixed point of a concrete semantic function
- Plan: define and compute a **sound abstract semantics** of the program