Program Analysis

5. Abstract Interpretation (1): Concrete Semantics

Kihong Heo

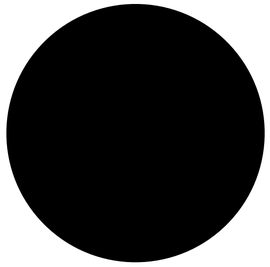
KAIST

Abstract Interpretation요약 해석

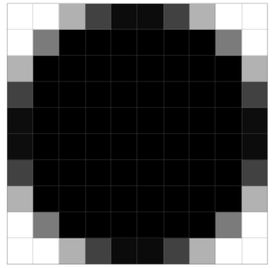
- A **powerful framework** for designing **correct** static analysis
  - Framework: given some inputs, a static analysis comes out
  - Powerful: all static analyses are understood in this framework
    (e.g., type systems, data-flow analysis, etc)
  - Correct: mathematically proven
- Established by Patrick 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

Purpose of Abstraction

"The purpose of abstraction is not to be vague,
but to create a new semantic level in which
one can be absolutely precise."

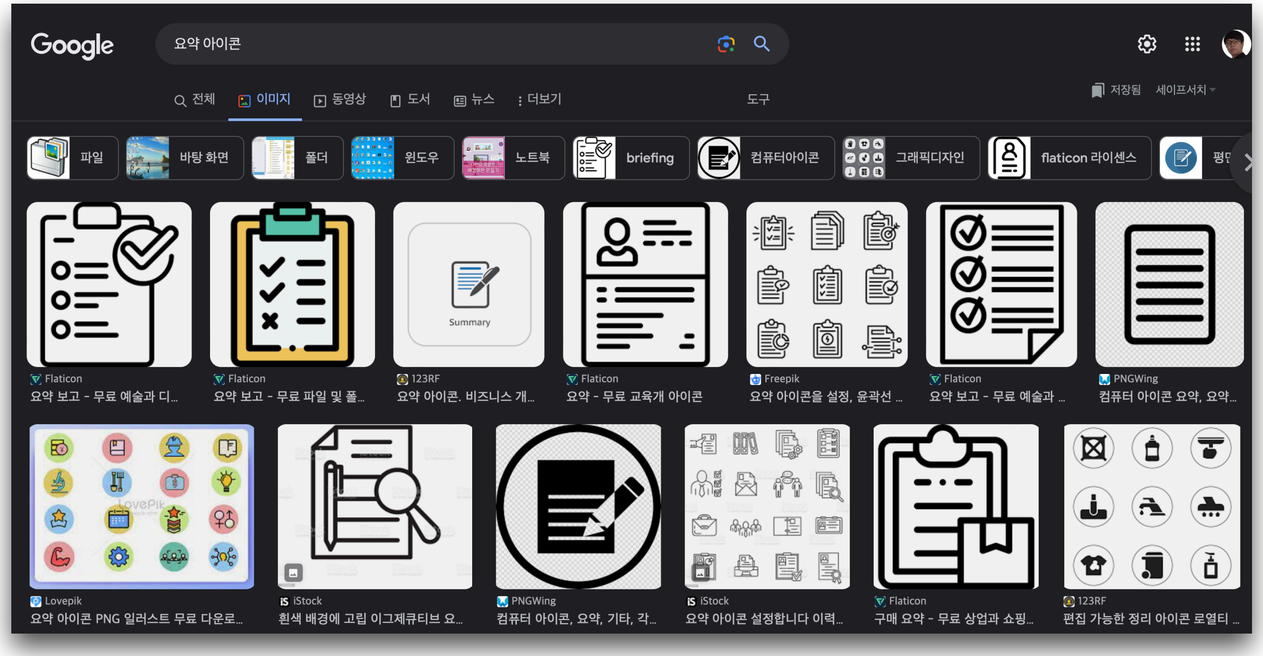
- Edsger W. Dijkstra

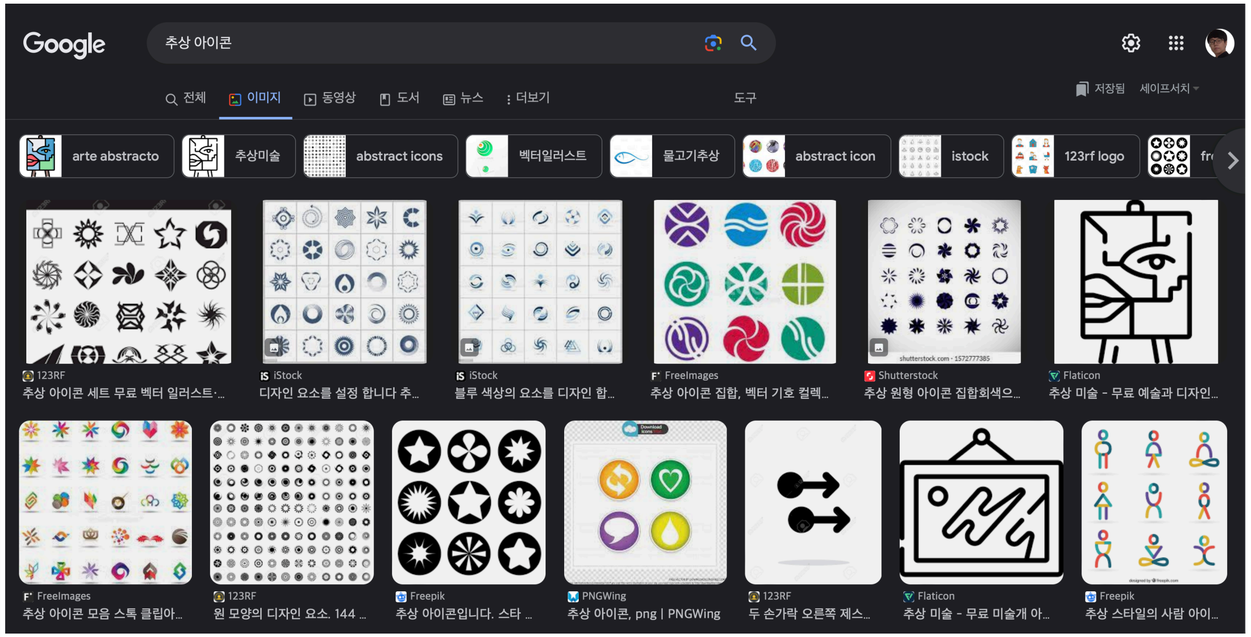
요약의 목적

"요약의 목적은 모호해지는 것이 아니라
완벽히 명료하게 의미를 전할 수 있는
새로운 수준을 만들어 내는 것이다."

- Edsger W. Dijkstra

요약 vs 추상


Google
요약 아이콘
전체
이미지
동영상
도서
뉴스
더보기
도구
저장됨
세이프서치
파일
바탕 화면
폴더
윈도우
노트북
briefing
컴퓨터아이콘
그래픽디자인
flaticon 라이센스
Summary
Flaticon
요약 보고 - 무료 예술과 디...
Flaticon
요약 보고 - 무료 파일 및 폴...
123RF
요약 아이콘. 비즈니스 개...
Flaticon
요약 - 무료 교육개 아이콘
Freepik
요약 아이콘을 설정, 윤곽선 ...
Flaticon
요약 보고 - 무료 예술과 ...
PNGWing
컴퓨터 아이콘 요약, 요약...
Lovepik
요약 아이콘 PNG 일러스트 무료 다운로...
iStock
흰색 배경에 고립 이그제큐티브 요...
PNGWing
컴퓨터 아이콘, 요약, 기타, 각...
iStock
요약 아이콘 설정합니다 이력...
Flaticon
구매 요약 - 무료 상업과 쇼핑...
123RF
편집 가능한 정리 아이콘 로열티 ...


Google
추상 아이콘
전체
이미지
동영상
도서
뉴스
더보기
도구
저장됨
세이프서치
arte abstracto
추상미술
abstract icons
벡터일러스트
물고기추상
abstract icon
istock
123rf logo
123RF
추상 아이콘 세트 무료 벡터 일러스트·...
iStock
디자인 요소를 설정 합니다 추...
iStock
블루 색상의 요소를 디자인 합...
Freelmages
추상 아이콘 집합, 벡터 기호 컬렉...
Shutterstock
추상 원형 아이콘 집합회색으...
Flaticon
추상 미술 - 무료 예술과 디자인...
Freelmages
추상 아이콘 모음 스톡 클립아...
123RF
원 모양의 디자인 요소. 144 ...
Freepik
추상 아이콘입니다. 스타 ...
PNGWing
추상 아이콘, png | PNGWing
123RF
두 손가락 오른쪽 제스...
Flaticon
추상 미술 - 무료 미술개 아...
Freepik
추상 스타일의 사람 아이...

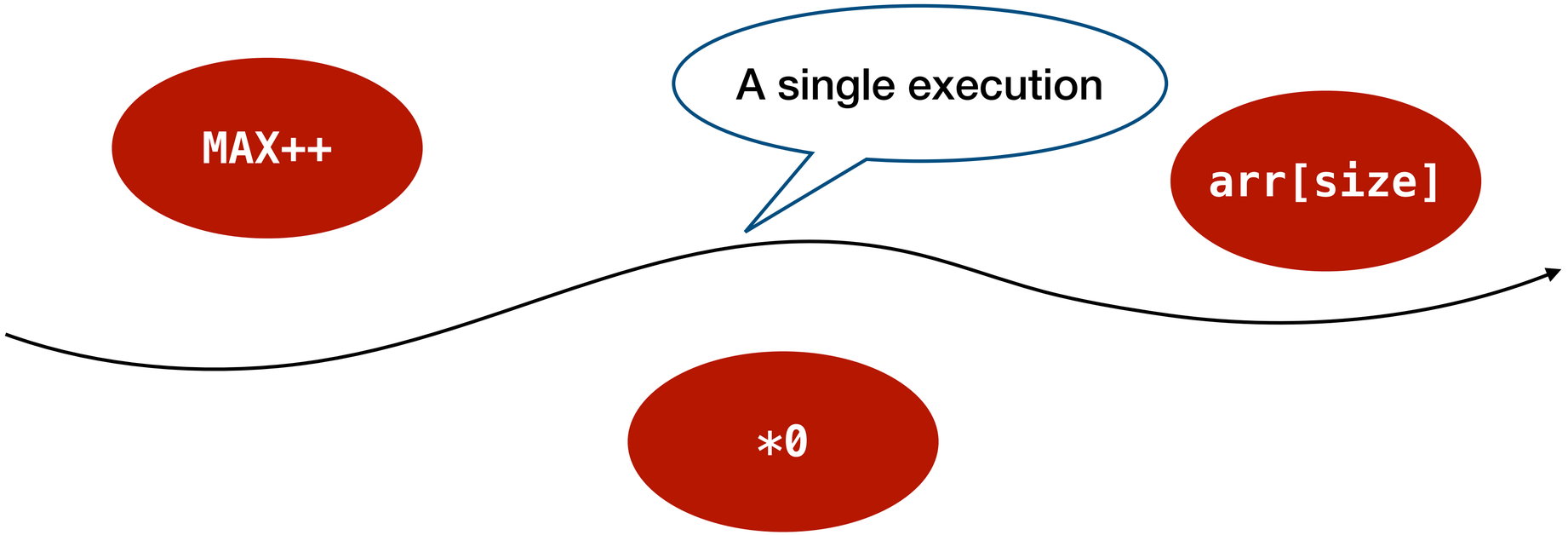
Example

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

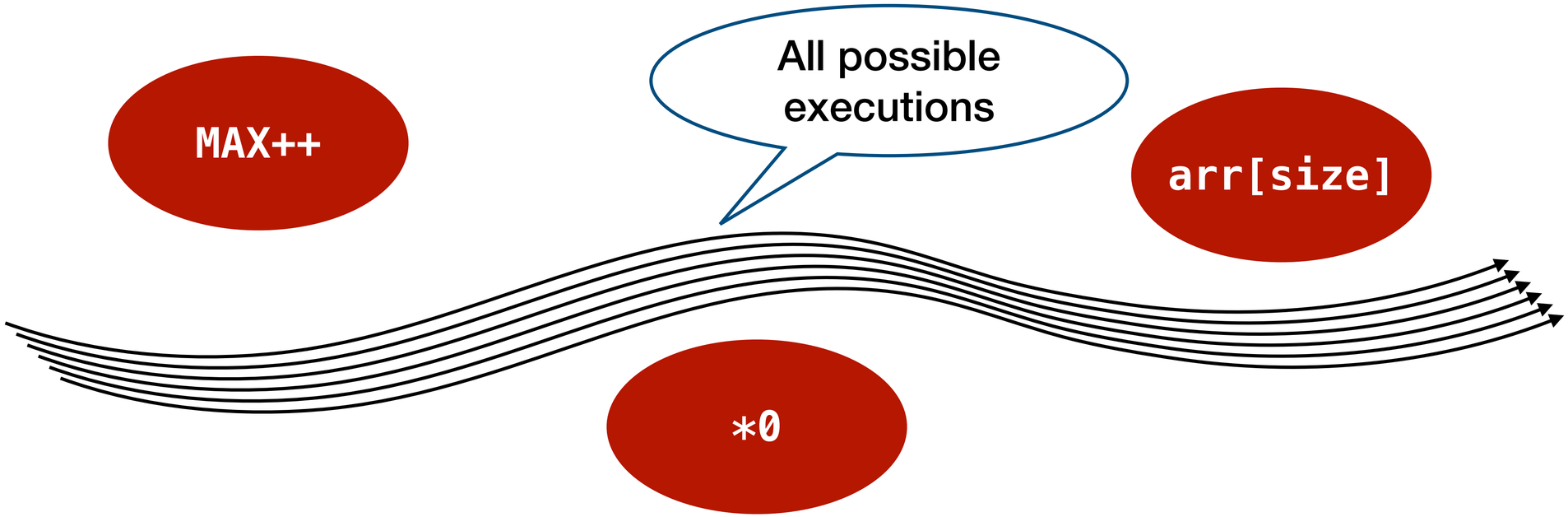
Q: What are the possible output values?

- 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)

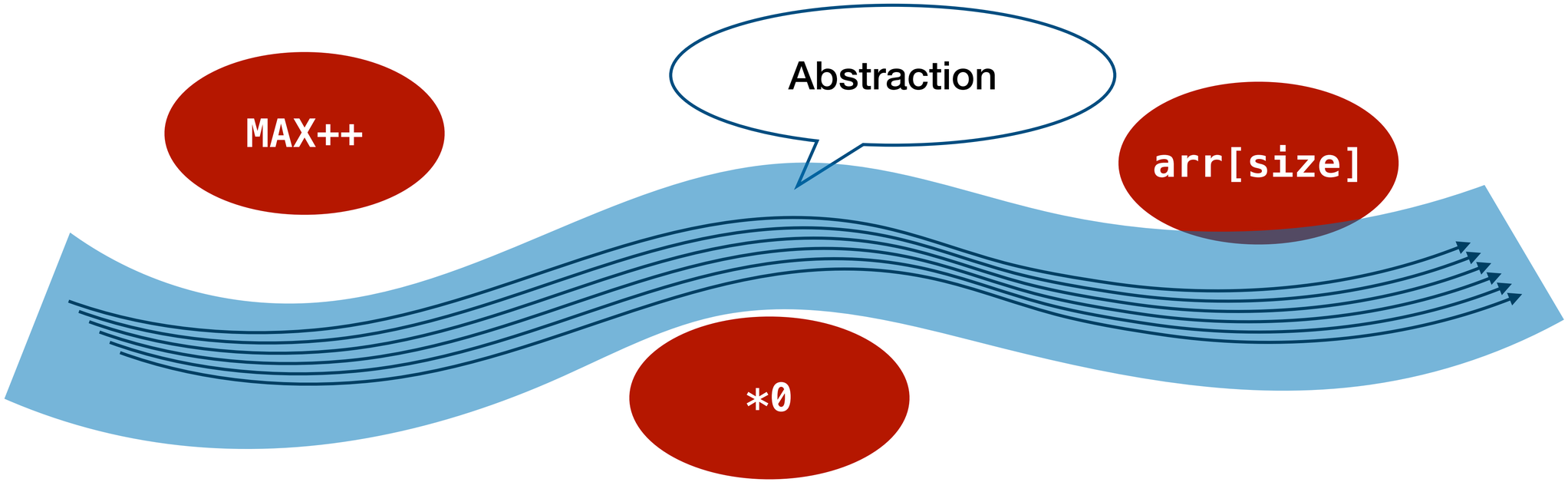
Abstraction of Executions



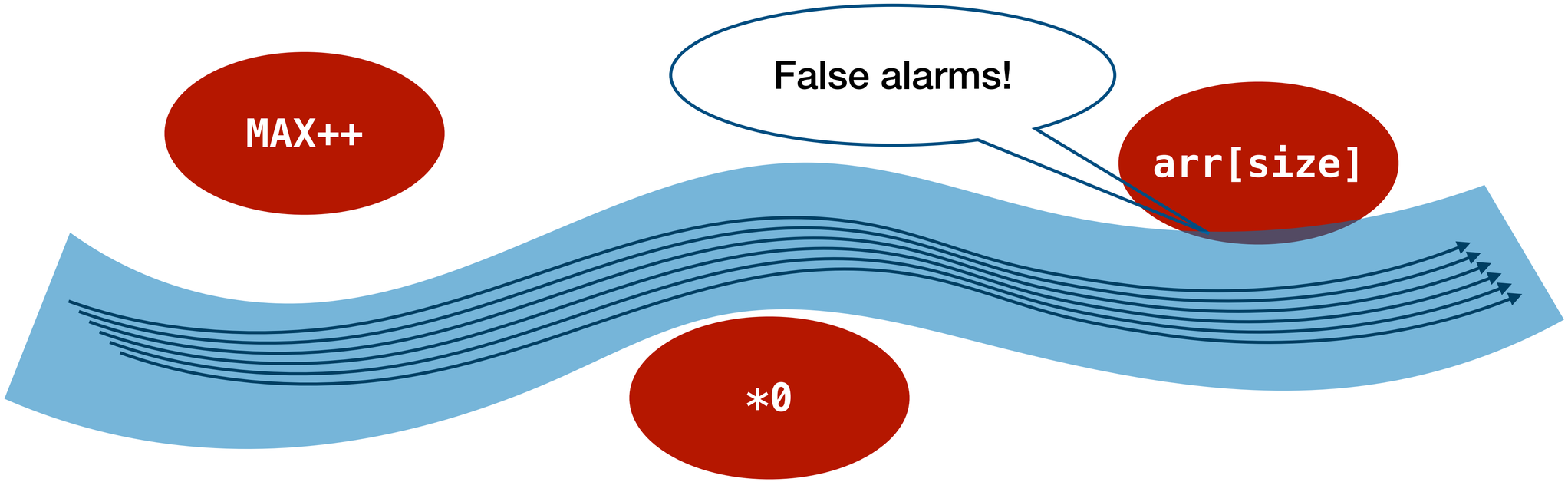
Abstraction of Executions



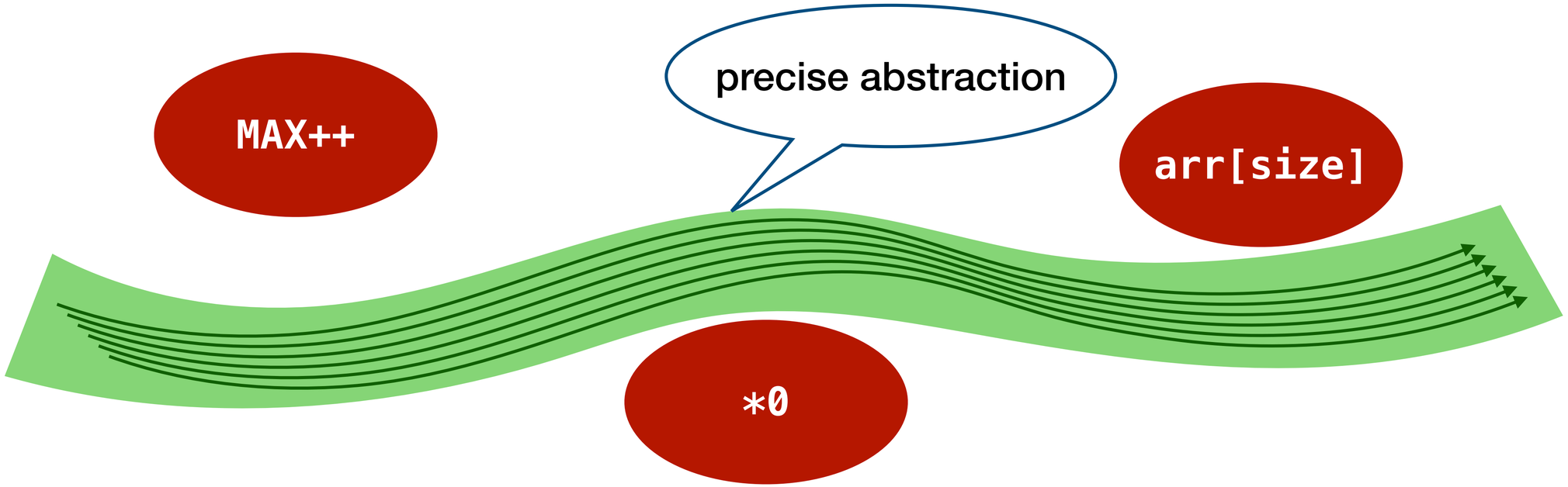
Abstraction of Executions



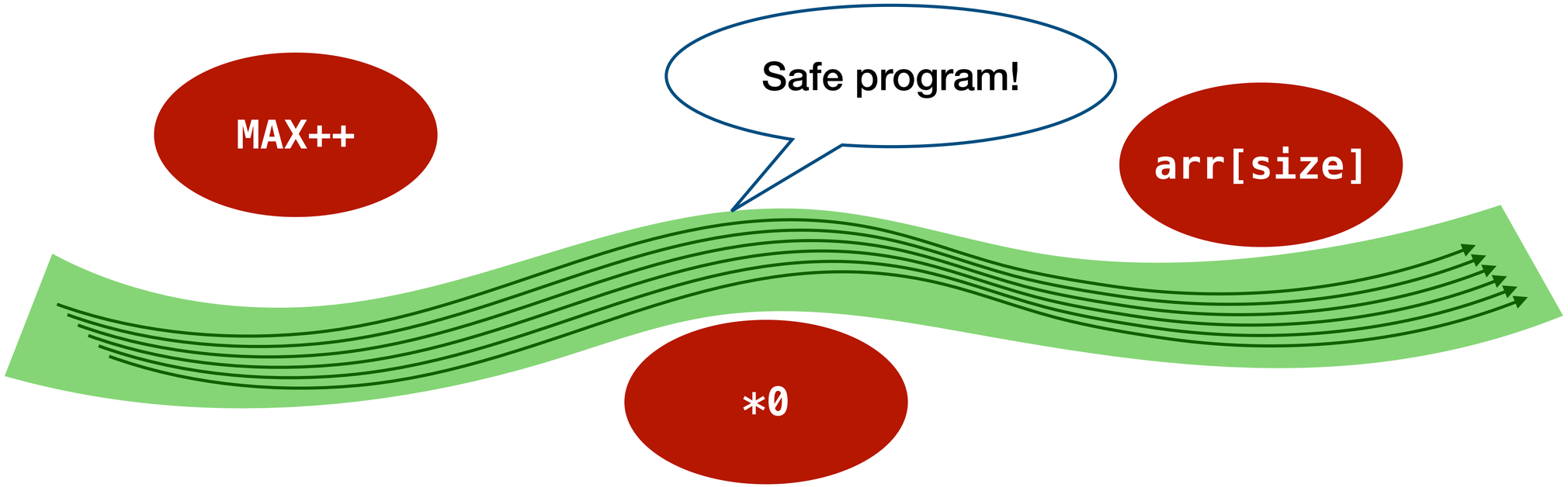
Abstraction of Executions



Abstraction of Executions



Abstraction of Executions



How to analyze?

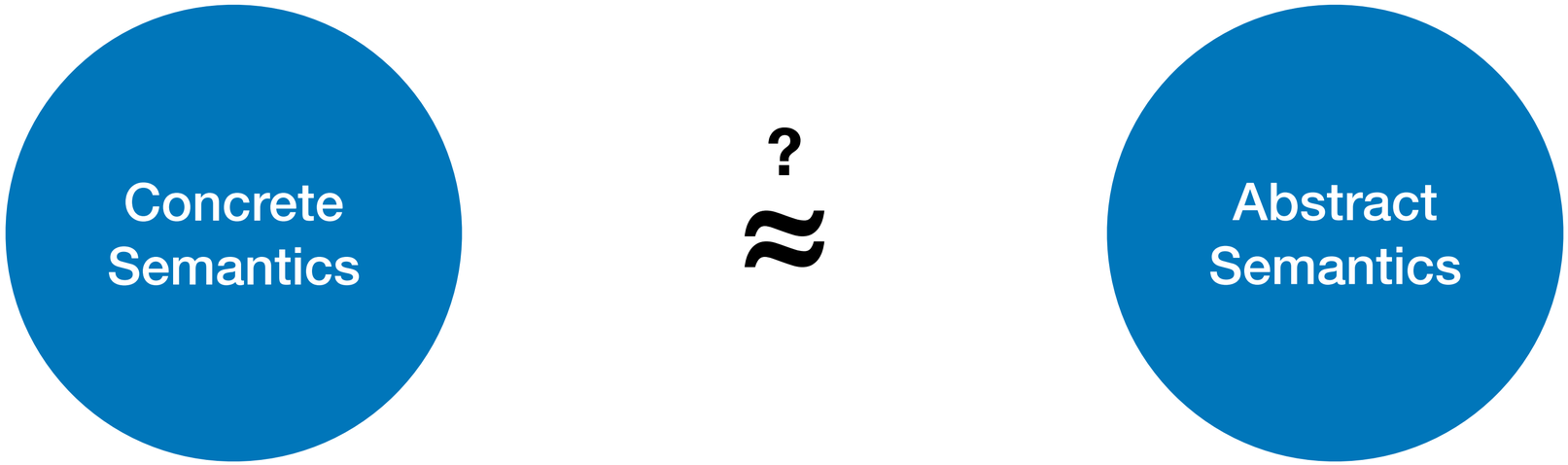
- Interpret the target program
  - with abstract semantics (= analyzer's concern)
  - not concrete semantics (= interpreter's and compiler's concern)

- Example

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

Concrete	Abstract 1	Abstract 2	Abstract 3
{3}	Int	Pos	PosOdd
{3, 5, 7, ...}	Int	Pos	PosOdd
{2, 4, 6, ...}	Int	Pos	PosEven

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: $[\![C]\!] = \mathsf{lfp} F \in \mathbb{D}$
  - Abstract semantics: $[\![C]\!]^\sharp = \bigsqcup_{i \geq 0} F^{\sharp i}(\bot) \in \mathbb{D}^\sharp$
- Requirements:
  - Relationship between $\mathbb{D}$ and $\mathbb{D}^\sharp$
  - Relationship between $F \in \mathbb{D} \to \mathbb{D}$ and $F^\sharp \in \mathbb{D}^\sharp \to \mathbb{D}^\sharp$
- Guarantees:
  - Correctness (soundness): $[\![C]\!] \approx [\![C]\!]^\sharp$
  - Computability: $[\![C]\!]^\sharp$ is computable within finite time

Design of Static Analysis

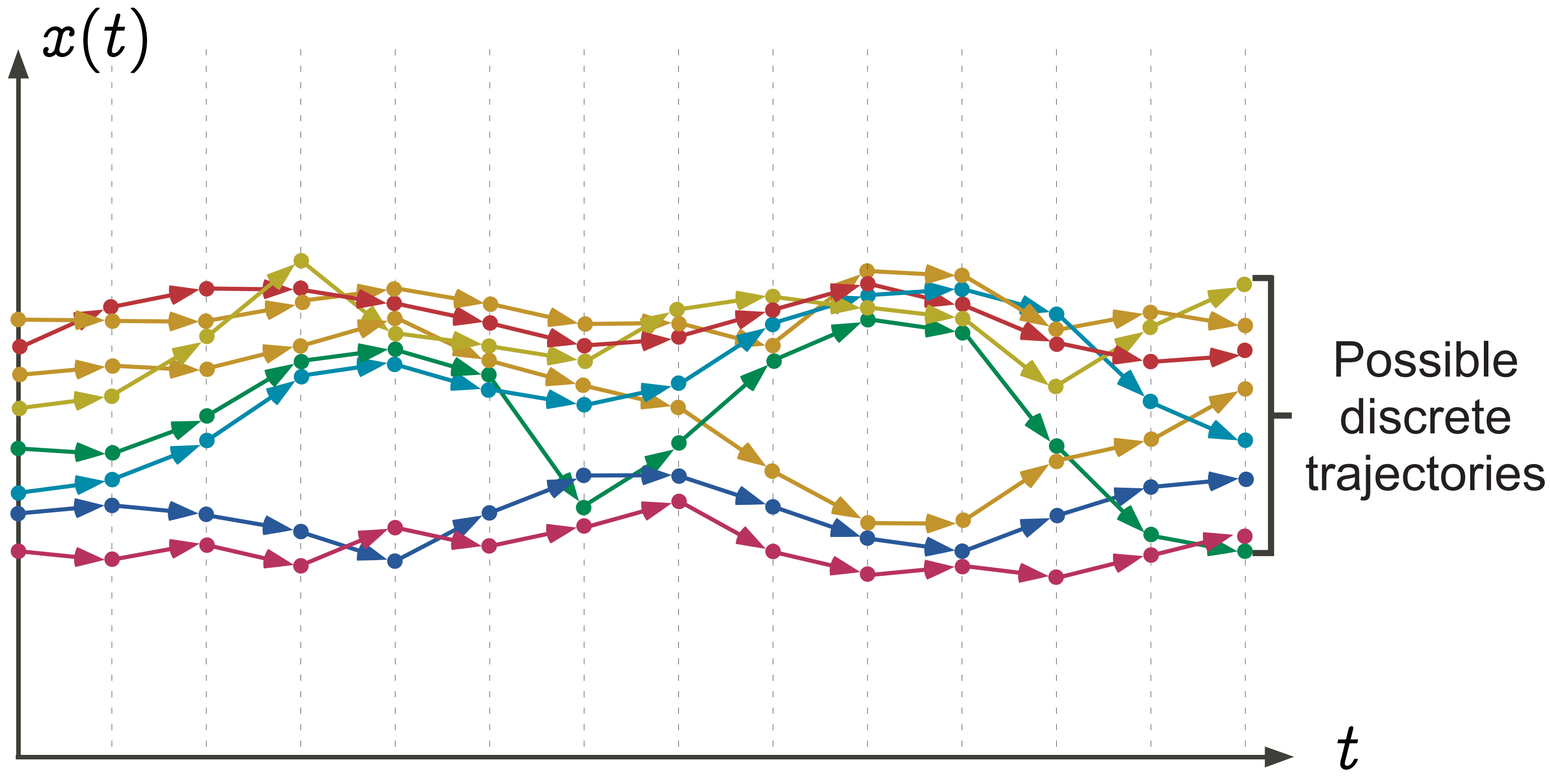
- Goal: conservative and terminating static analysis
- Design principles:
  - Define concrete semantics
  - Define abstract semantics (sound w.r.t the concrete semantics)
- Computation & implementation:
  - Abstract semantics of a program: the least fixed point of the semantic function
  - Static analyzer: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

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

Define Standard Semantics



*from Patrick Cousot's slides

Standard Semantics

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

$$\mathbf{lfp}F = \bigsqcup_{i \geq 0} F^i(\bot)$$

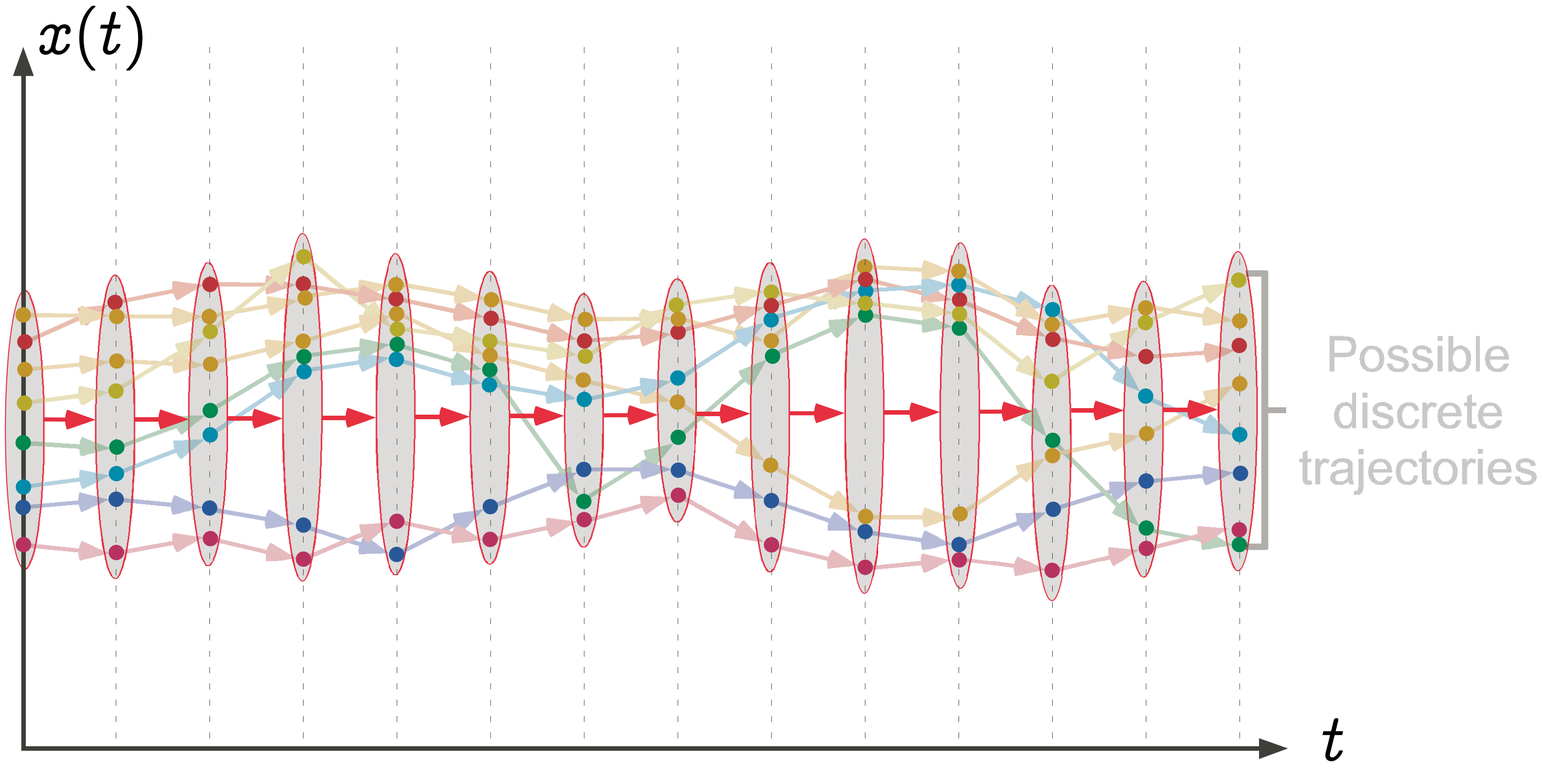
Standard Semantics of Commands

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

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

Transitions of Sets of States

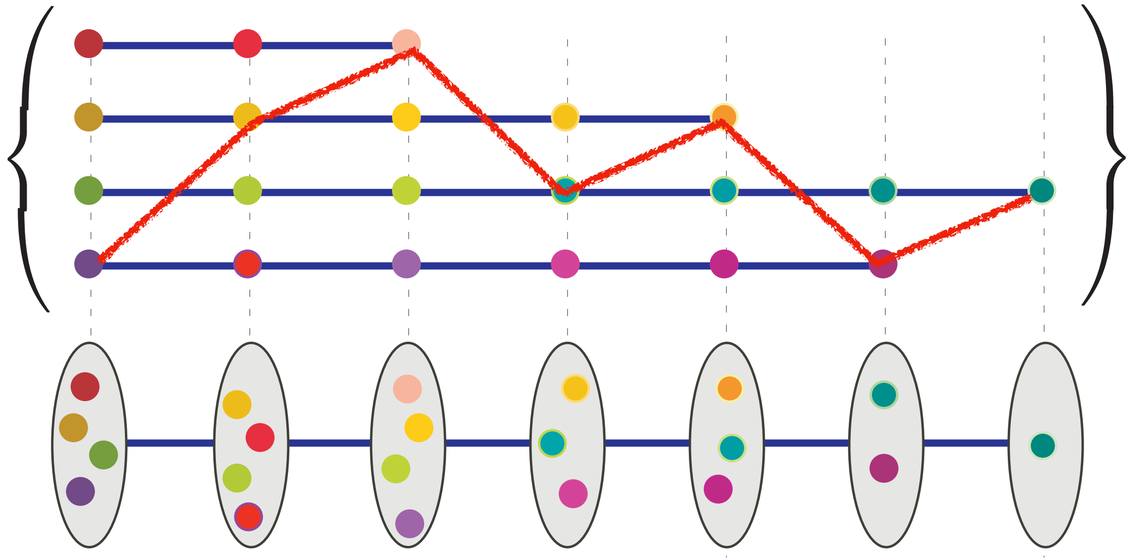


*from Patrick Cousot's slides

Traces vs Reachable States

Traces

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

$$\mathbf{lfp}F = \bigsqcup_{i \geq 0} F^i(\bot)$$

Example

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

Concrete Semantics of Expressions

$$
\begin{aligned}
[\![E]\!]_{\wp} &: \wp(\mathbb{M}) \to \wp(\mathbb{Z}) \\
[\![n]\!]_{\wp} &= \lambda M.\{n\} \\
[\![x]\!]_{\wp} &= \lambda M.\{m(x) \mid m \in M\} \\
[\![E_1 \odot E_2]\!]_{\wp} &= \lambda M.\{[\![E_1]\!](m) \odot [\![E_2]\!](m) \mid m \in M\}
\end{aligned}
$$

$$
\begin{aligned}
[\![B]\!]_{\wp} &: \wp(\mathbb{M}) \to \wp(\mathbb{M}) \\
[\![\mathsf{true}]\!]_{\wp} &= \lambda M.M \\
[\![\mathsf{false}]\!]_{\wp} &= \lambda M.\emptyset \\
[\![E_1 \oslash E_2]\!]_{\wp} &= \lambda M.\{m \in M \mid [\![E_1]\!](m) \oslash [\![E_2]\!](m) = \mathsf{true}\}
\end{aligned}
$$

Concrete Semantics of Commands

$$[\![C]\!]_{\wp} \;:\; \wp(\mathbb{M}) \to \; \wp(\mathbb{M})$$

$$\begin{aligned}
[\![\texttt{skip}]\!]_{\wp} &= \lambda M.M \\
[\![C_0\,\texttt{;}\,C_1]\!]_{\wp} &= \lambda M.[\![C_1]\!]_{\wp} \circ [\![C_0]\!]_{\wp}(M) \\
[\![x\,\texttt{:=}\,E]\!]_{\wp} &= \lambda M.\{m\{x \mapsto v\} \mid v \in [\![E]\!](m), m \in M\} \\
[\![\texttt{input}(x)]\!]_{\wp} &= \lambda M.\{m\{x \mapsto n\} \mid m \in M, n \in \mathbb{Z}\} \\
[\![\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!]_{\wp} &= \lambda M.[\![C_1]\!]_{\wp} \circ [\![B]\!]_{\wp}(M) \cup [\![C_2]\!]_{\wp} \circ [\![\neg B]\!]_{\wp}(M) \\
[\![\texttt{while}\ B\ C]\!]_{\wp} &= \lambda M.[\![\neg B]\!]_{\wp}\big(\mathbf{lfp}\lambda X.M \cup [\![C]\!]_{\wp} \circ [\![B]\!]_{\wp}(X)\big)
\end{aligned}$$

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