Introduction to Information Security

13. Static Analysis

Kihong Heo

KAIST

Recap

"Program testing can be used to show the presence of bugs, but never to show their absence!"

- Edsger W. Dijkstra

ACM - Association for Computing Machinery

9분 ·

#ACMTuringAward recipient Edsger Wybe Dijkstra was born on this day in 1930. Dijkstra received the 1972 ACM A.M. Turing Award for fundamental contributions to programming as a high, intellectual challenge; for eloquent insistence and practical demonstration that programs should be composed correctly, not just debugged into correctness; for illuminating perception of problems at the foundations of program design.

https://bit.ly/3nVC3ux

Static Analysis정적 분석

- **Over-approximate** (not exact) all possible behavior of a program
- In general, **sound and automatic, but incomplete**
  - May have spurious results
- Based on a foundational theory : Abstract interpretation
- Variants:
  - Under-approximating static analysis: automatic, **complete**, but **unsound**
  - Bug finder: automatic, **unsound**, **incomplete**, and heuristics

Example: Buffer Overrun

```
static char *curfinal = “HDACB  FE”;
keysym = read_from_input();

if ((KeySym)(keysym) >= 0xFF9987)
{
    unparseputc((char)(keysym – 0xFF91 + ‘P’), pty);
    key = 1;
}
else if (keysym >= 0)
{
    if (keysym < 16)
    {
        if (read_from_input())
        {
            if (keysym >= 10) return;
            curfinal[keysym] = 1;
        }
        else
        {
            curfinal[keysym] = 2;
        }
    }
    if (keysym < 10)
        unparseput(curfinal[keysym], pty);
}
```

curfinal: buffer of size 10

keysym : any integer

keysym: [0, 15]

SAFE

keysym: [0, 9]

Buffer-overrun

size of curfinal: [10, 10]
keysym: [0, 15]

keysym: [0, 9]

SAFE

Example: SQL Injection

```
1:  char *input = input();
2:  char *san = NULL;
3:
4:  if (input != NULL) {
5:    bool cond = unknown();
6:    if (cond) {
7:      san = sanitize(input);
8:      sql_query(san);
9:    } else {
10:     san = input;
11:   }
12: sql_query(san);
13: }
```

input: MaybeVul

san: NotVul

cond: Unknown

san: NotVul

SAFE

san: MaybeVul

SQL Injection

san: MaybeVul

Success Stories

Domain-specific Verification

General-purpose Bug-finding

SLAM

Windows Device Driver

Microsoft

Astrée

Airbus Controller

ENS / AbsInt

coverity®

Stanford / Synopsys

Infer

Facebook

Sparrow

SNU / Fasoo.com

PolySpace TECHNOLOGIES

Mathworks

CODESONAR

GrammaTech

lgtm.com by Semmle™

Semmle / Github

JULIA

JuliaSoft

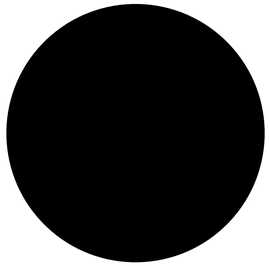
GCC

GCC

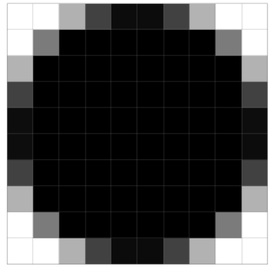
LLVM/Clang

Abstraction요약

- 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

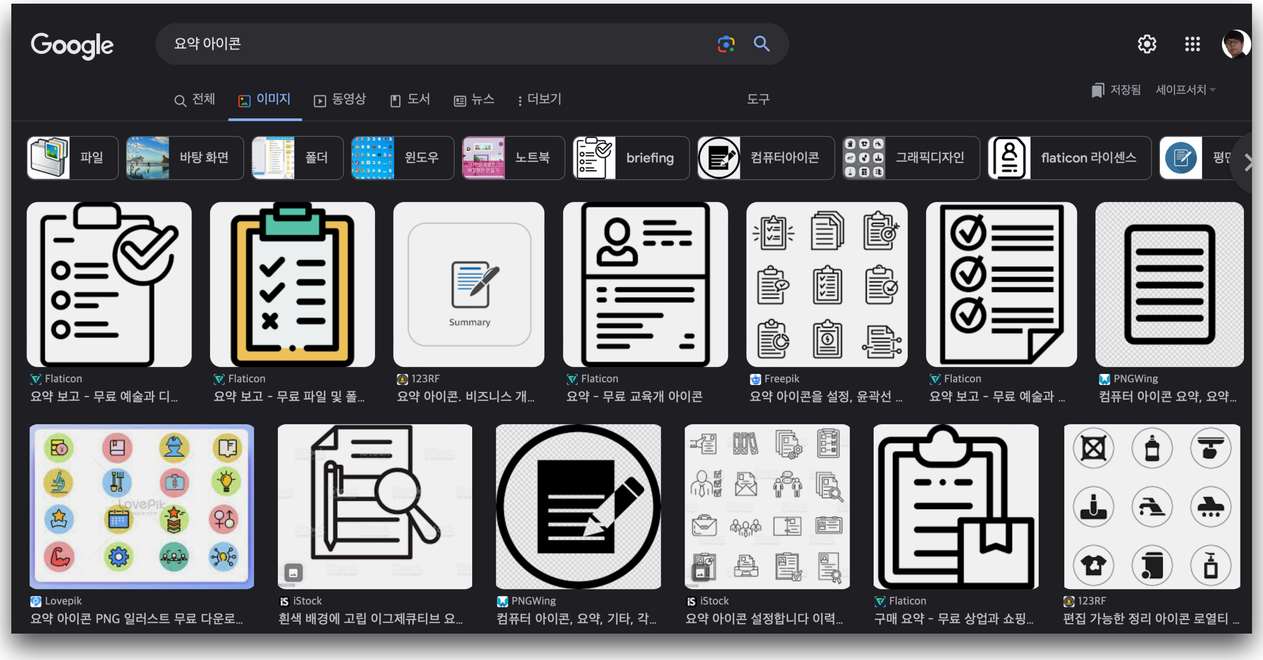
"The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise."

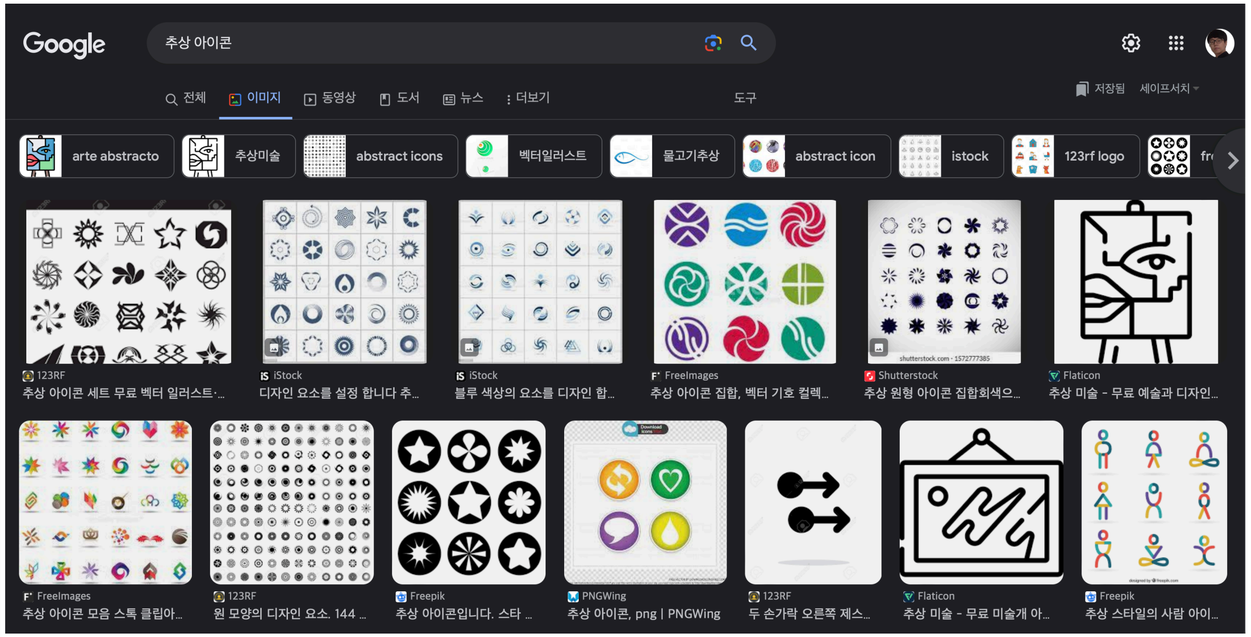
- Edsger W. Dijkstra

"요약의 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 of Abstrations

```
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)

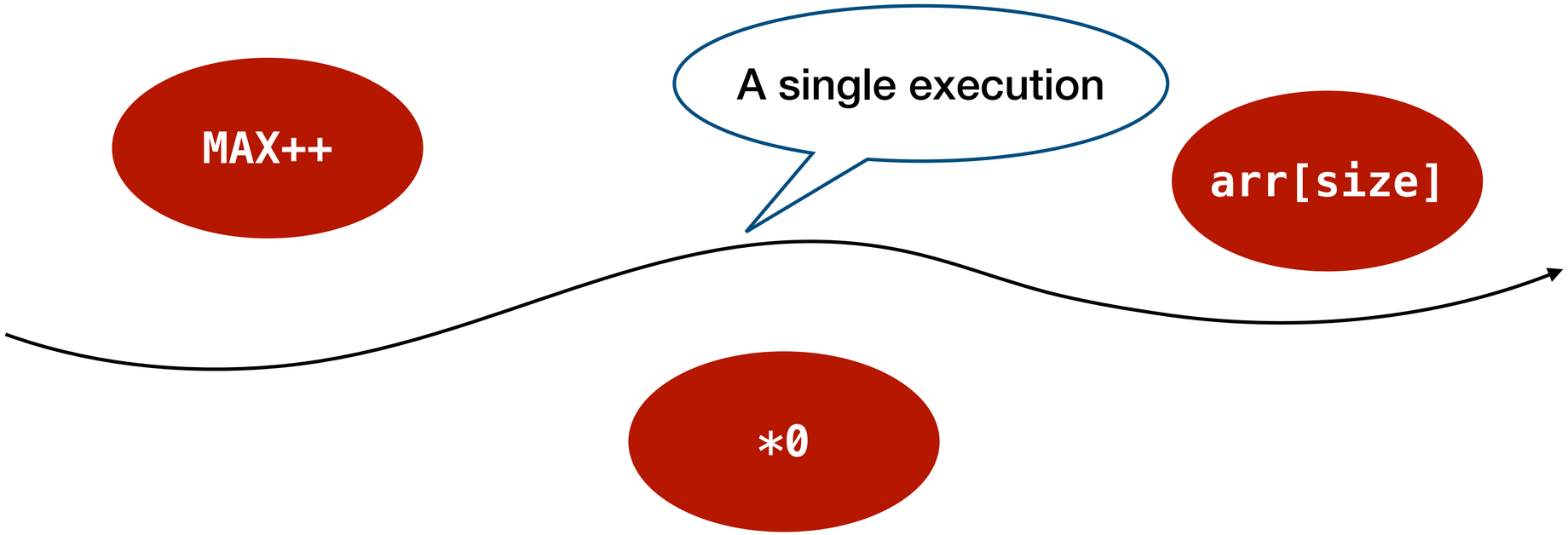
False Positives

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

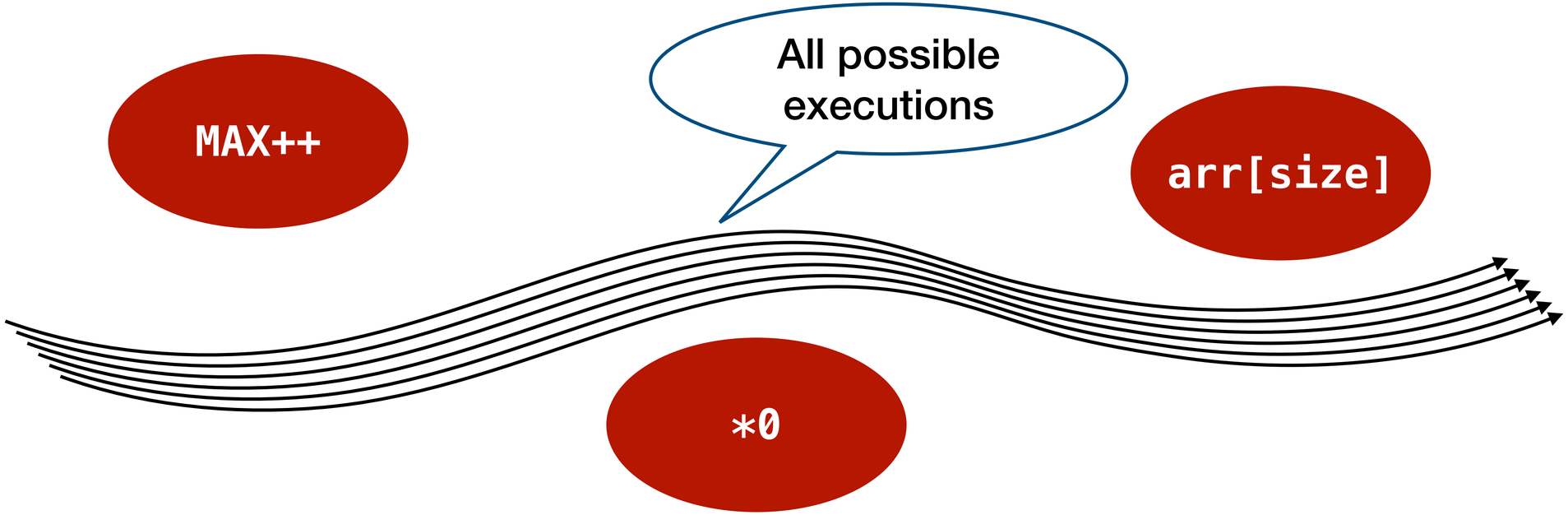
Q: Is the assertion always true?

- Concrete interpretation : …
- Abstract interpretation 1 : “Maybe not, because x is an integer” (false positive)
- Abstract interpretation 2 : “Maybe not, because x is a positive integer” (false positive)
- Abstract interpretation 3 : “Yes, because x is a positive even integer” (correct)

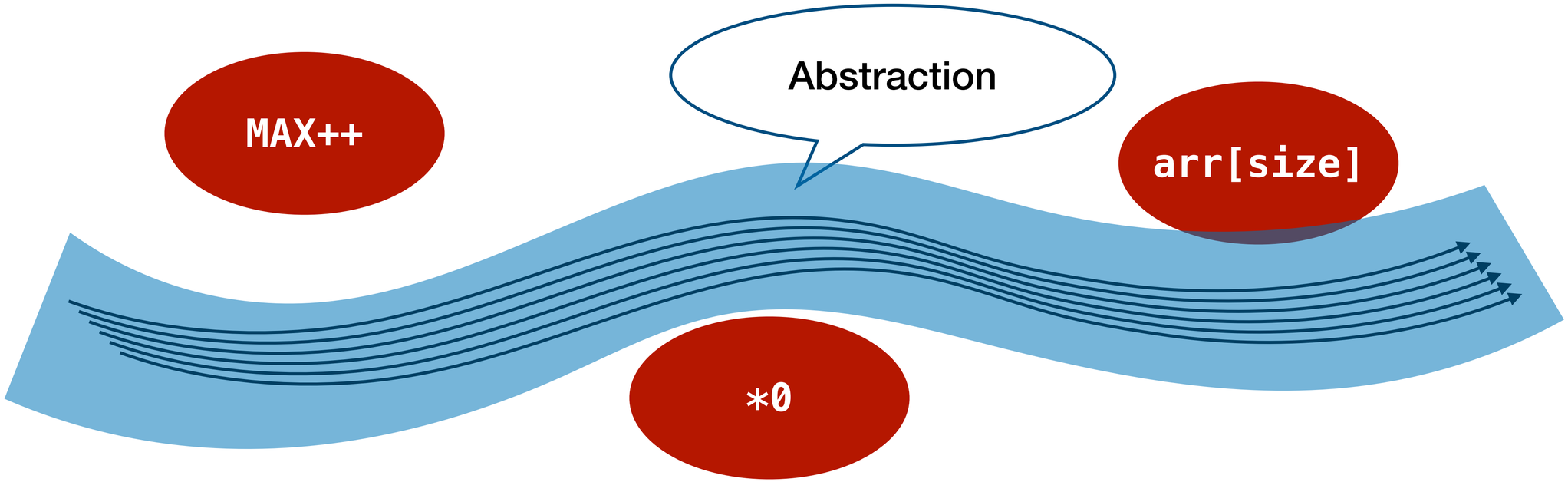
Abstraction of Executions



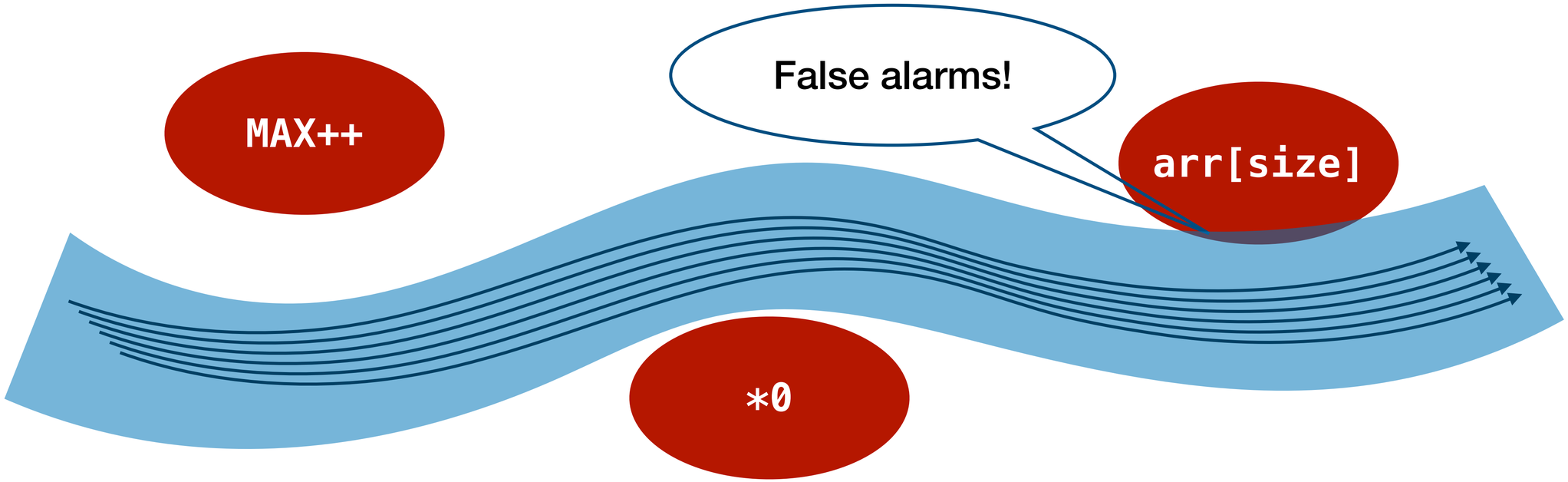
Abstraction of Executions



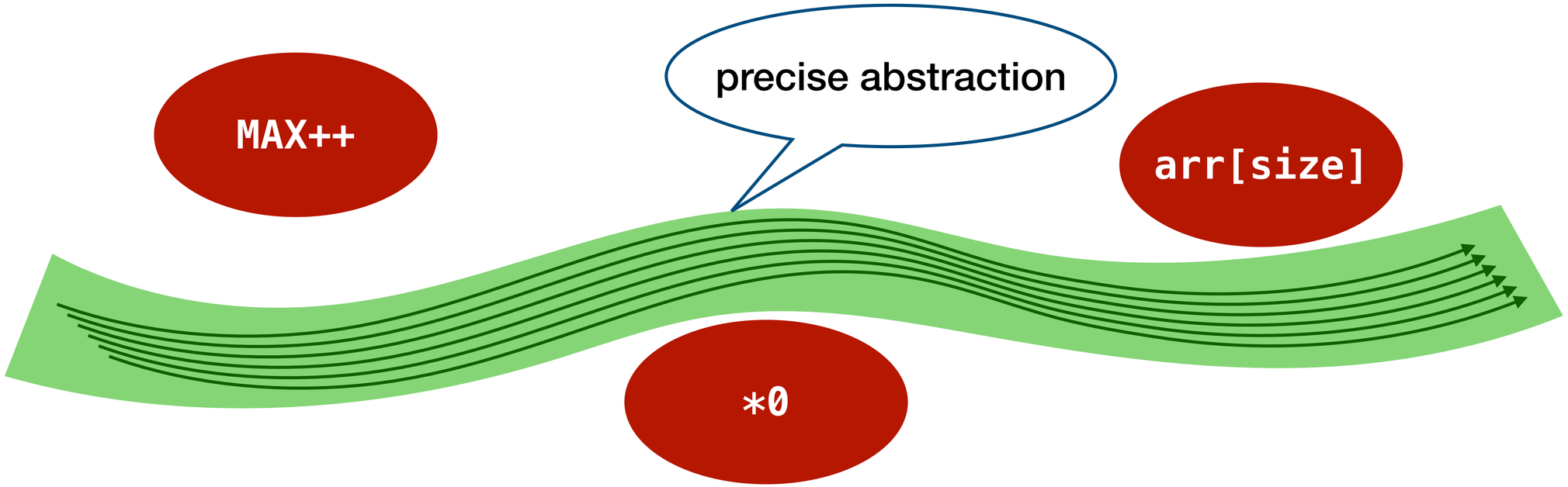
Abstraction of Executions



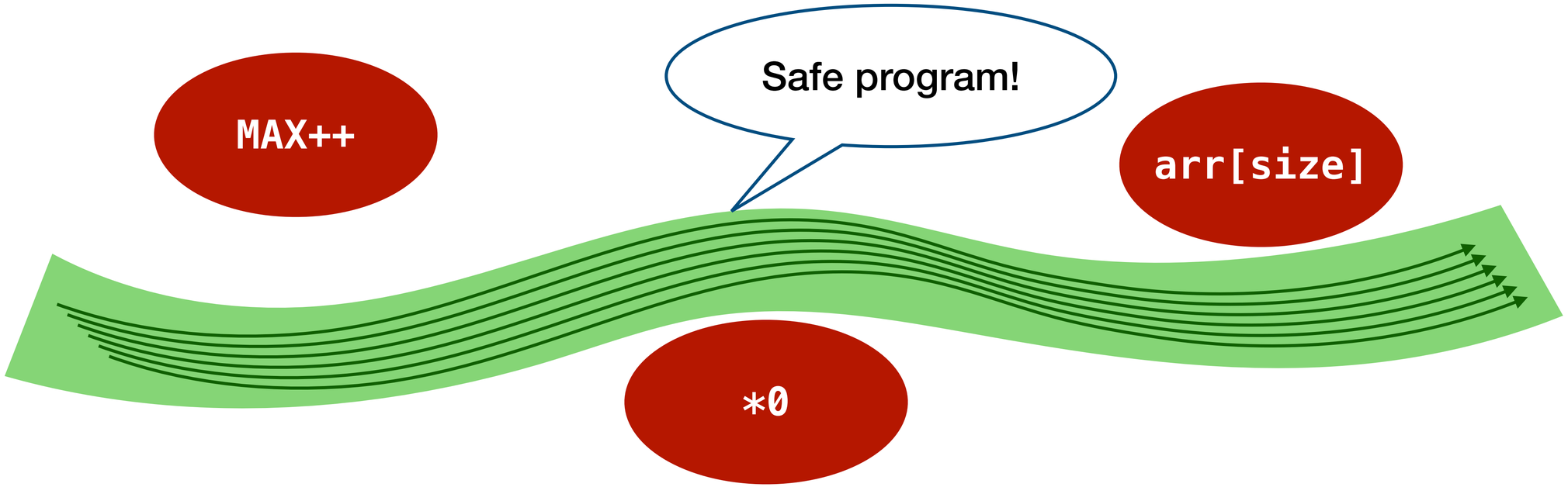
Abstraction of Executions



Abstraction of Executions



Abstraction of Executions



How to analyze?

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

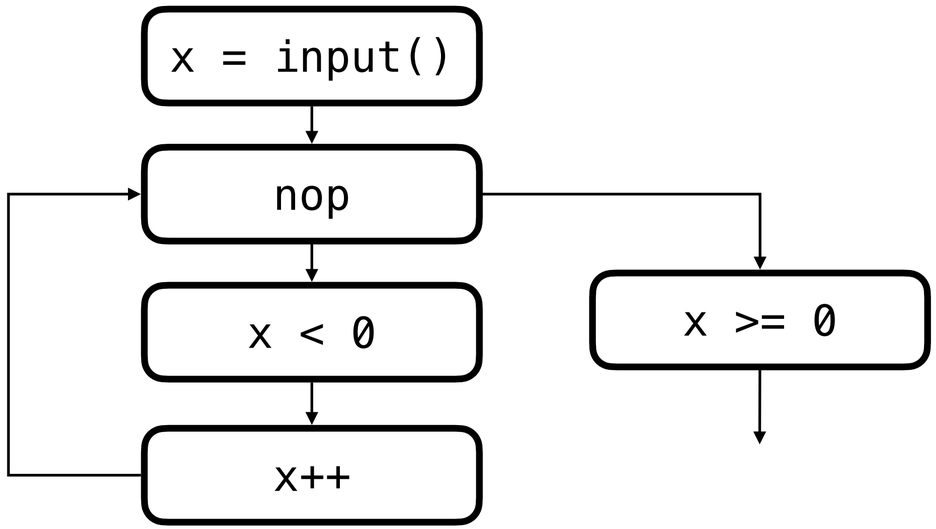
```
x = 3;
while (*) {
  x += 2;
}
x -= 1;
print(x);
```

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

Program as a Graph

- Control-flow graph: a directed graph $G = (Node, \rightarrow)$
  - Nodes: atomic statements or conditions
  - Edges: execution order between the statements

```
x = input();
while (x < 0) {
  x++;
}
```



Inference Rules추론 규칙

- Methods to **derive new facts from known facts** (i.e., derivable = provable)
- Notation of inference rule:

$$\frac{P_1 \quad \cdots \quad P_n}{P} \quad \begin{array}{l}\textit{premise(s)} \\ \textit{conclusion}\end{array} \qquad \frac{human(x) \implies mortal(x) \qquad human(s)}{mortal(s)}$$

- Inference rule with zero premise is called an **axiom**

- Examples: inference rules of propositional logic

$$\frac{}{\text{true}} \quad \frac{A \qquad B}{A \wedge B} \quad \frac{A \wedge B}{A} \quad \frac{A \wedge B}{B} \quad \frac{A}{A \vee B} \quad \frac{B}{A \vee B} \quad \frac{A \qquad A \implies B}{B} \quad \ldots$$

Inductive Definition of Set (1)

- Induction: a way to define a set (as well as a proof technique) using inference rules
  - E.g., natural number, list, binary tree, expression, etc
- Consider function ϕ derived from the set Φ of inference rules

$$\phi(Y) = \{x \mid \frac{X}{x} \in \Phi, X \subseteq Y\}$$

- Example:
  - $\phi(\{\})$, $\phi(\{0\})$, $\phi(\{3,5,7\})$

Inductive Definition of Set (2)

- Closure (닫힘): a set is **closed under an operation** if performing that operation on members of the set always produces a member of that set
- Inductively defined set: the least closed set of ϕ (i.e., the least fixed point of ϕ)

$$S = \bigcap \{X \mid \phi(X) \subseteq X\}$$

- How to obtain such a set?

$$\phi^0 \cup \phi^1 \cup \phi^2 \cup \cdots = \bigcup_{i \in \mathbb{N}} \phi^i$$

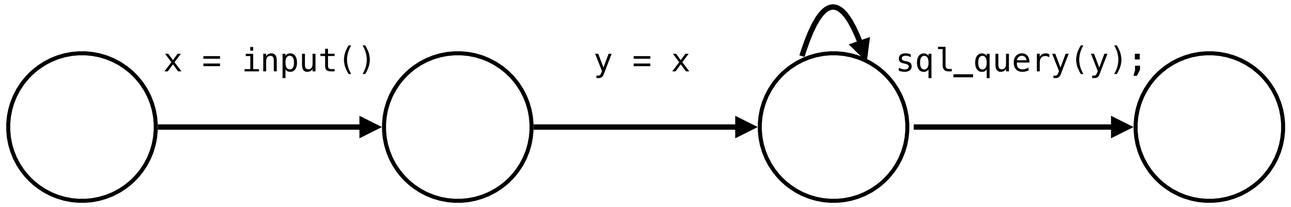
- Example
  - $\mathbb{N}$
  - list

Closure in Computer Science

- Transitive closure of a graph = all reachable nodes from each node

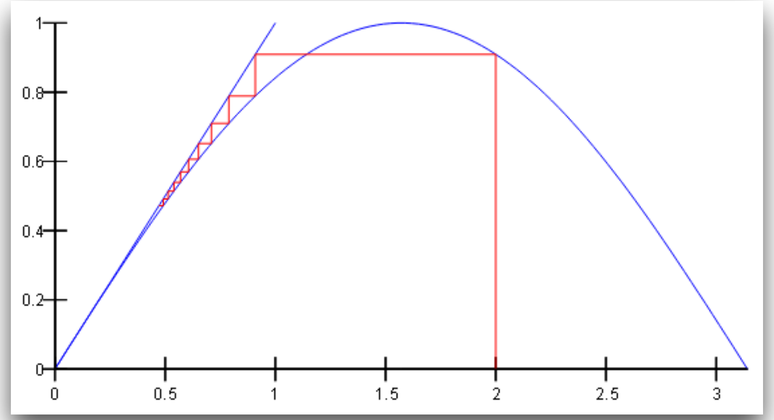
$$\frac{x \rightarrow y}{x \rightarrow^* y} \qquad \frac{x \rightarrow y \qquad y \rightarrow^* z}{x \rightarrow^* z}$$

- Closure in static analysis = estimating all possible effects of the operations



Static Analysis by Closure

- Static analysis result = a closure of rules = a fixed point of rules
- How to compute a fixed point?
- Fixed point iteration:
  1. Start from the initial facts
  2. Apply rules and collect more facts
  3. Repeat until no more new facts are generated



Case Study 1: Pointer Analysis

- Goal: the set of locations that each pointer may store globally
  - Computing "points-to" facts between two variables
  - $a \rightarrow b$: "a points to b"
- Example:

$$
\begin{array}{lcl}
C & \rightarrow & L := R \\
 & | & C; C \\
 & | & \texttt{while}\ B\ C \\
L & \rightarrow & x \mid *x \\
R & \rightarrow & x \mid *x \mid \&x \\
B & & \text{boolean expressions}
\end{array}
$$

```
x = &a;
y = &x;
while (*) {
  *y = &b;
}
*x = *y
```

All possible points-to facts

```
{x→a, y→x, x→b,
a→a, b→b, a→b, b→a}
```

Rules for Pointer Analysis

- The rule has the following form $$\frac{C \quad i_1 \cdots i_k}{j}$$
  - Premises: the program has component C and the current set has $i_1 \cdots i_k$
  - Conclusion: j is derivable (so, add j to the set)

Base case
(collecting initial facts)

$$\frac{x := \&y}{x \to y}$$

Inductive cases
(deriving new facts)

$$\frac{x := y \quad y \to z}{x \to z} \qquad \frac{*x := y \quad x \to w \quad y \to z}{w \to z} \qquad \frac{*x := \&y \quad x \to w}{w \to y}$$

$$\frac{x := *y \quad y \to z \quad z \to w}{x \to w} \qquad \frac{*x := *y \quad x \to w \quad y \to z \quad z \to v}{w \to v}$$

Example

```
x = &a;
y = &x;
while (*) {
  *y = &b;
}
*x = *y
```

$$\frac{x := \&y}{x \to y}$$

$$\frac{x := y \quad y \to z}{x \to z} \qquad \frac{*x := y \quad x \to w \quad y \to z}{w \to z} \qquad \frac{*x := \&y \quad x \to w}{w \to y}$$

$$\frac{x := *y \quad y \to z \quad z \to w}{x \to w} \qquad \frac{*x := *y \quad x \to w \quad y \to z \quad z \to v}{w \to v}$$

Collect the initial facts directly from program text

{x→a, y→x}

Example

```
x = &a;
y = &x;
while (*) {
  *y = &b;
}
*x = *y
```

$$\frac{x := \&y}{x \to y}$$

$$\frac{x := y \quad y \to z}{x \to z} \qquad \frac{*x := y \quad x \to w \quad y \to z}{w \to z} \qquad \boxed{\frac{*x := \&y \quad x \to w}{w \to y}}$$

$$\frac{x := *y \quad y \to z \quad z \to w}{x \to w} \qquad \frac{*x := *y \quad x \to w \quad y \to z \quad z \to v}{w \to v}$$

Apply the rules to the known facts

{x→a, y→x, x→b}

Example

```
x = &a;
y = &x;
while (*) {
  *y = &b;
}
*x = *y
```

$$\frac{x := \&y}{x \to y}$$

$$\frac{x := y \quad y \to z}{x \to z} \qquad \frac{*x := y \quad x \to w \quad y \to z}{w \to z} \qquad \frac{*x := \&y \quad x \to w}{w \to y}$$

$$\frac{x := *y \quad y \to z \quad z \to w}{x \to w} \qquad \boxed{\frac{*x := *y \quad x \to w \quad y \to z \quad z \to v}{w \to v}}$$

Repeat until reaching a fixed point

{x→a, y→x, x→b, a→a, b→b, a→b, b→a}

Case Study 2: Taint Analysis

- Goal: find an unsanitized information flow from a source point to a sink point
  - Source: read untrustworthy inputs (taint)
  - Sanitizer: neutralize malicious inputs
  - Sink: safety- / security-critical points
- Compute def-use facts between two variables
  - $a \rightarrow b$: “the value of a is used to define the value of b w/o sanitization”
- Applications: format string attack, code injection attack, etc

Example

```
1:  char *input = input();
2:  char *san = NULL;
3:
4:  if (input != NULL) {
5:    bool cond = unknown();
6:    if (cond) {
7:      san = sanitize(input);
8:      sql_query(san);
9:    } else {
10:     san = input;
11:   }
12:   sql_query(san);
13:}
```

Source

Sanitizer

Sink

Sink

Rules for Taint Analysis

Base cases
(collecting initial facts)

$$\frac{l : x := E}{\mathsf{Def}(l, x)} \quad \frac{l : E := y}{\mathsf{Use}(l, y)} \quad \frac{l : x := \texttt{source}()}{\mathsf{Src}(l)} \quad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{San}(l)} \quad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{Use}(l, y)} \quad \frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Sink}(l)} \quad \frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Use}(l, x)}$$

Inductive cases
(deriving new facts)

$$\frac{\mathsf{Def}(l_1, x) \quad \mathsf{Use}(l_2, x)}{\mathsf{Edge}(l_1, l_2)} \quad \frac{\mathsf{Src}(l_1) \quad \mathsf{Edge}(l_1, l_2)}{\mathsf{Path}(l_1, l_2)} \quad \frac{\mathsf{Path}(l_1, l_2) \quad \neg\mathsf{San}(l_2) \quad \mathsf{Edge}(l_2, l_3)}{\mathsf{Path}(l_1, l_3)} \quad \frac{\mathsf{Path}(l_1, l_2) \quad \mathsf{Sink}(l_2)}{\mathsf{Alarm}(l_1, l_2)}$$

Example

```
1: x = source();
2: y = x;
3: if(*) {
4:   z = sanitize(y);
5: } else {
6:   sink(y);
7: }
8: sink(z);
```

$$\frac{l : x := E}{\mathsf{Def}(l, x)} \quad \frac{l : E := y}{\mathsf{Use}(l, y)} \quad \frac{l : x := \texttt{source}()}{\mathsf{Src}(l)}$$

$$\frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{San}(l)} \quad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{Use}(l, y)}$$

$$\frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Sink}(l)} \quad \frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Use}(l, x)}$$

Collect the initial facts directly from program text

{ Def(1,x), Src(1), Def(2,y), Use(2,x) **}**

Example

```
1: x = source();
2: y = x;
3: if(*) {
4:   z = sanitize(y);
5: } else {
6:   sink(y);
7: }
8: sink(z);
```

$$\frac{l : x := E}{\mathsf{Def}(l, x)} \quad \frac{l : E := y}{\mathsf{Use}(l, y)} \quad \frac{l : x := \texttt{source}()}{\mathsf{Src}(l)}$$

$$\frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{San}(l)} \quad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{Use}(l, y)}$$

$$\frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Sink}(l)} \quad \frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Use}(l, x)}$$

Collect the initial facts directly from program text

{ Def(1,x), Src(1), Def(2,y), Use(2,x),
Def(4,z), Use(4,y), San(4) **}**

Example

```
1: x = source();
2: y = x;
3: if(*) {
4:   z = sanitize(y);
5: } else {
6:   sink(y);
7: }
8: sink(z);
```

$$\frac{l : x := E}{\mathsf{Def}(l, x)} \quad \frac{l : E := y}{\mathsf{Use}(l, y)} \quad \frac{l : x := \texttt{source}()}{\mathsf{Src}(l)}$$

$$\frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{San}(l)} \quad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{Use}(l, y)}$$

$$\frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Sink}(l)} \quad \frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Use}(l, x)}$$

Collect the initial facts directly from program text

{ Def(1,x), Src(1), Def(2,y), Use(2,x),
Def(4,z), Use(4,y), San(4),
Sink(6), Use(6,y) **}**

Example

```
1: x = source();
2: y = x;
3: if(*) {
4:   z = sanitize(y);
5: } else {
6:   sink(y);
7: }
8: sink(z);
```

$$\frac{l : x := E}{\mathsf{Def}(l, x)} \quad \frac{l : E := y}{\mathsf{Use}(l, y)} \quad \frac{l : x := \texttt{source}()}{\mathsf{Src}(l)}$$

$$\frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{San}(l)} \quad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{Use}(l, y)}$$

$$\frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Sink}(l)} \quad \frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Use}(l, x)}$$

Collect the initial facts directly from program text

{ Def(1,x), Src(1), Def(2,y), Use(2,x),
Def(4,z), Use(4,y), San(4),
Sink(6), Use(6,y),
Sink(8), Use(8,z)**}**

Example

```
1: x = source();
2: y = x;
3: if(*) {
4:   z = sanitize(y);
5: } else {
6:   sink(y);
7: }
8: sink(z);
```

$$\boxed{\frac{\mathsf{Def}(l_1, x) \quad \mathsf{Use}(l_2, x)}{\mathsf{Edge}(l_1, l_2)}} \qquad \frac{\mathsf{Src}(l_1) \quad \mathsf{Edge}(l_1, l_2)}{\mathsf{Path}(l_1, l_2)} \qquad \frac{\mathsf{Path}(l_1, l_2) \quad \neg\mathsf{San}(l_2) \quad \mathsf{Edge}(l_2, l_3)}{\mathsf{Path}(l_1, l_3)}$$

$$\frac{\mathsf{Path}(l_1, l_2) \quad \mathsf{Sink}(l_2)}{\mathsf{Alarm}(l_1, l_2)}$$

Apply the rules to the known facts

{ Def(1,x), Src(1), Def(2,y), Use(2,x),
Def(4,z), Use(4,y), San(4),
Sink(6), Use(6,y),
Sink(8), Use(8,z),
Edge(1,2), Edge(2,4), Edge(2,6)**}**

Example

```
1: x = source();
2: y = x;
3: if(*) {
4:   z = sanitize(y);
5: } else {
6:   sink(y);
7: }
8: sink(z);
```

$$\frac{\mathsf{Def}(l_1, x) \quad \mathsf{Use}(l_2, x)}{\mathsf{Edge}(l_1, l_2)} \qquad \frac{\mathsf{Src}(l_1) \quad \mathsf{Edge}(l_1, l_2)}{\mathsf{Path}(l_1, l_2)} \qquad \frac{\mathsf{Path}(l_1, l_2) \quad \neg\mathsf{San}(l_2) \quad \mathsf{Edge}(l_2, l_3)}{\mathsf{Path}(l_1, l_3)}$$

$$\frac{\mathsf{Path}(l_1, l_2) \quad \mathsf{Sink}(l_2)}{\mathsf{Alarm}(l_1, l_2)}$$

Repeat until reaching a fixed point

{ Def(1,x), Src(1), Def(2,y), Use(2,x),
Def(4,z), Use(4,y), San(4),
Sink(6), Use(6,y),
Sink(8), Use(8,z),
Edge(1,2), Edge(2,4), Edge(2,6),
... Alarm(1,6)**}**

Termination of Static Analysis

- Static analysis always terminates even when the target program does not
- Does the previous static analysis terminate?
- Why?

Summary

- Static analysis: over-approximate all possible behavior of programs
- Soundness and incompleteness of static analysis:
  - Able to prove the absence of bugs but may have spurious error reports
- Example: static analysis by monotonic closure
  - Static analysis = setting up initial facts + collecting new facts by rules (until a fixed point)
- Challenges (covered in CS524):
  - Soundness proof
  - Precise and scalable static analysis for complicated properties