CS348
Static Analysis

Min Suk Kang **& Kihong Heo**

KAIST

Recap

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

- Edsger W. Dijkstra

Static Analysis

- **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

```
char *input = input();
char *san = NULL;

if (input != NULL) {
  bool cond = unknown();
  if (cond) {
    san = sanitize(input);
    sql_query(san);
  } else {
    san = input;
  }
  sql_query(san);
}
```

input: MaybeVul

san: NotVul

cond: Unknown

san: NotVul

SAFE

san: MaybeVul

SQL Injection

san: MaybeVul

False Positives

- May have spurious alarms because of incompleteness
  - Can be improved by more advanced design

```
1: int x = ...
2: char *arr = malloc(x);
3: int i = x - 1;
4: arr[i] = 0;
```

x: [1, 5]

size of arr: [1, 5]

i: [0, 4]

Buffer Overrun?

```
1: char *input = input();
2: char *san = NULL;
3:
5: if (input != NULL) {
4:   bool cond = unknown();
6:   if (cond) {
7:     san = sanitize(input);
8:     sql_query(san);
9:   } else {
10:    san = input;
11:  }
12:
13:  if (cond) {
     sql_query(san);
     }
16:}
```

input: MaybeVul

san: NotVul

cond: Unknown

san: NotVul

san: MaybeVul

cond: Unknown

san: MaybeVul

SQL Injection?

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

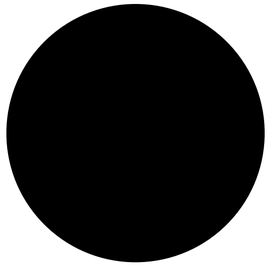
GCC

GCC

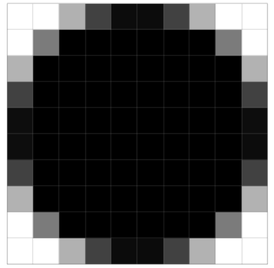
LLVM/Clang

Abstraction

- 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

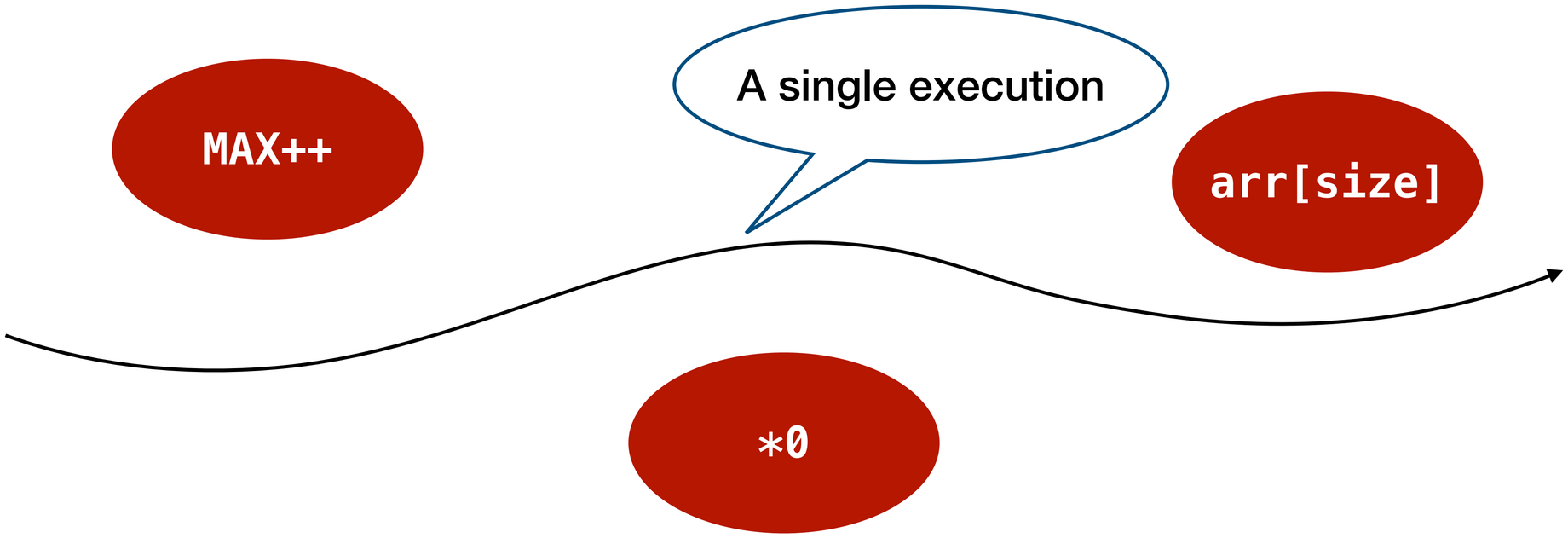
Example

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

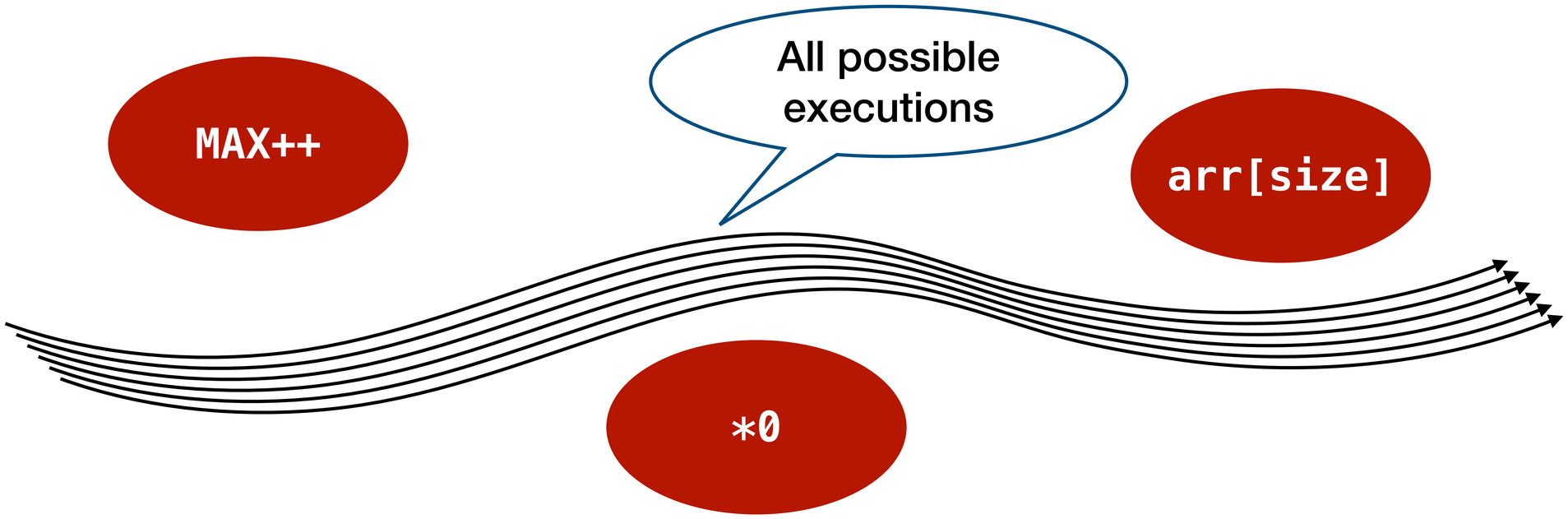
Q: What are the possible output values?

- 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)

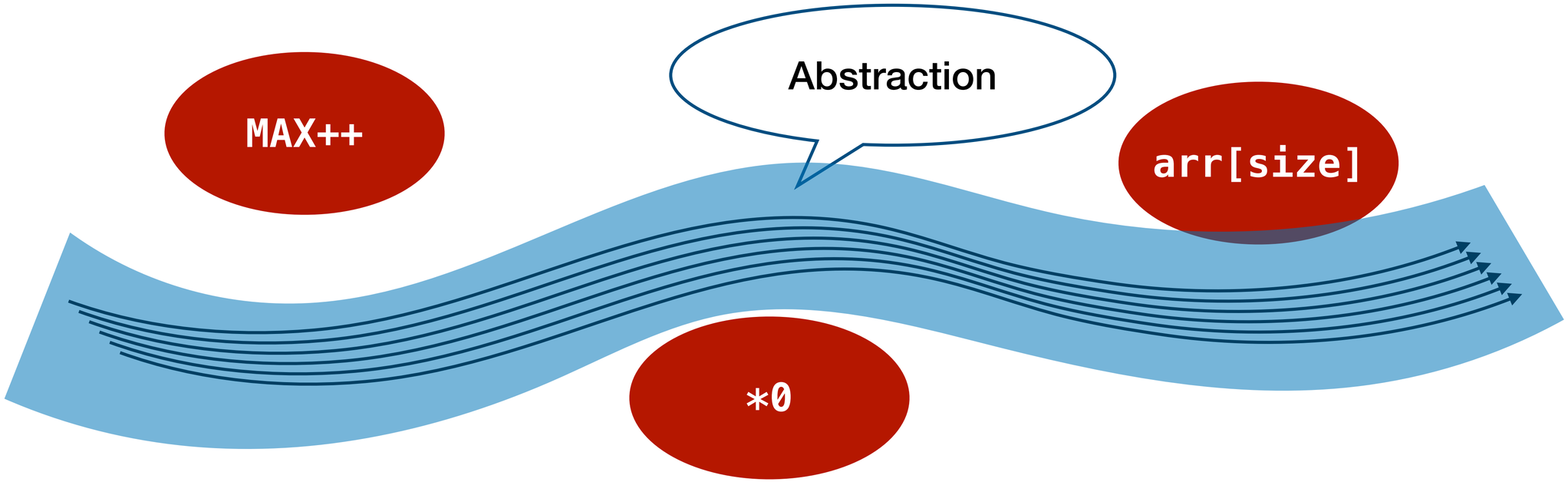
Abstraction of Executions



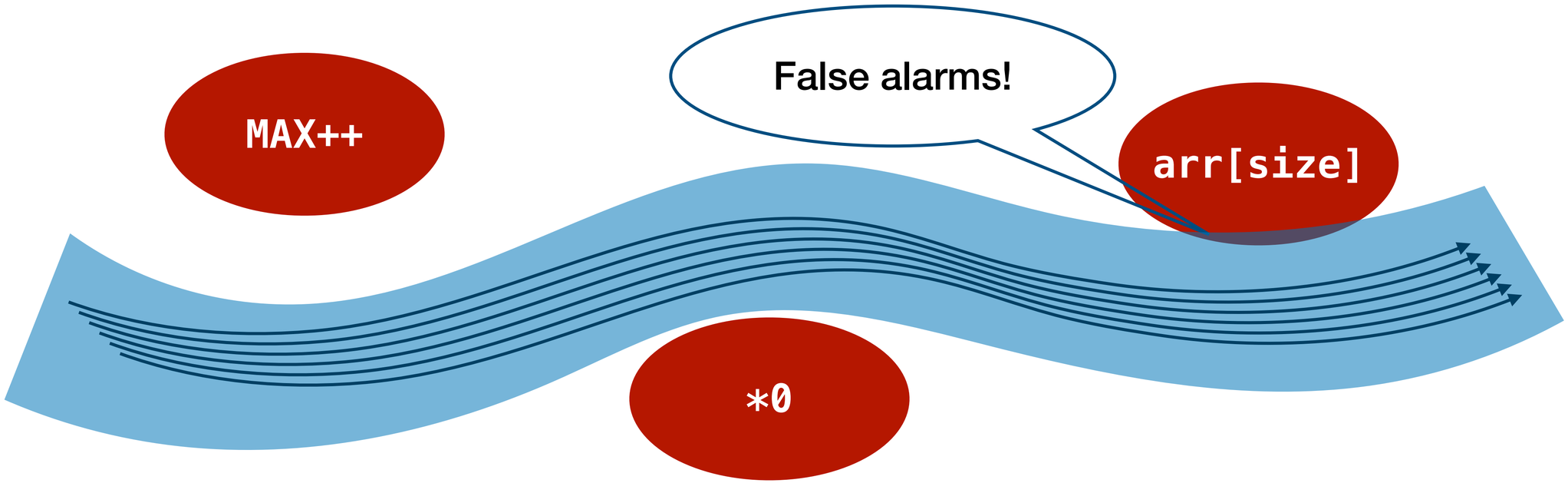
Abstraction of Executions



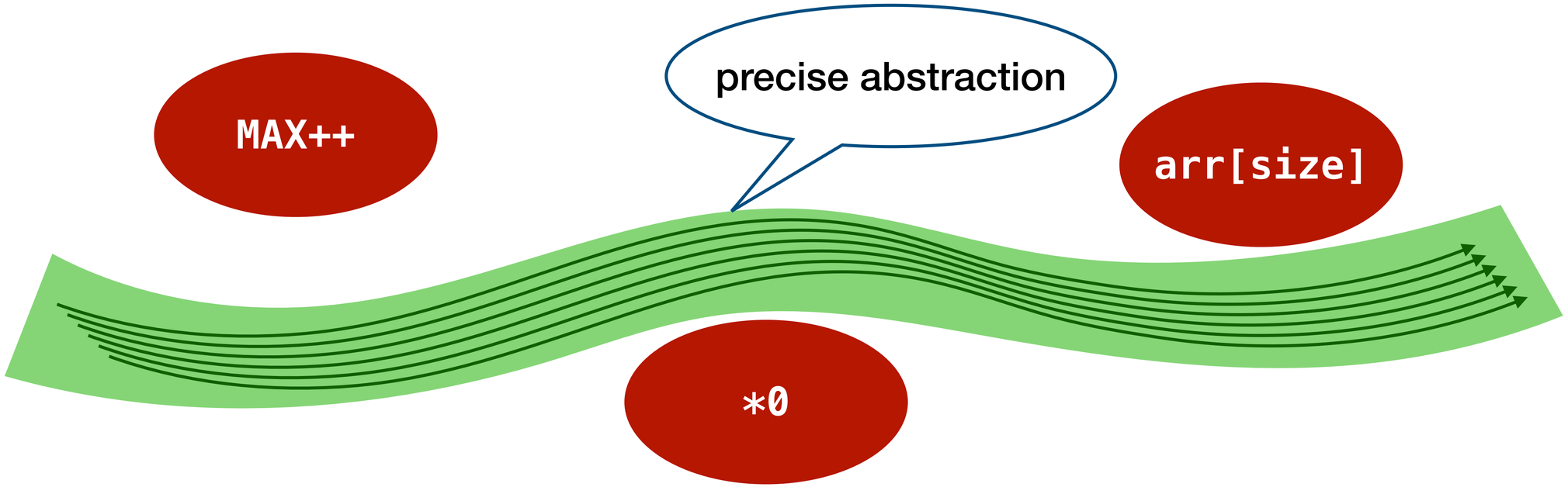
Abstraction of Executions



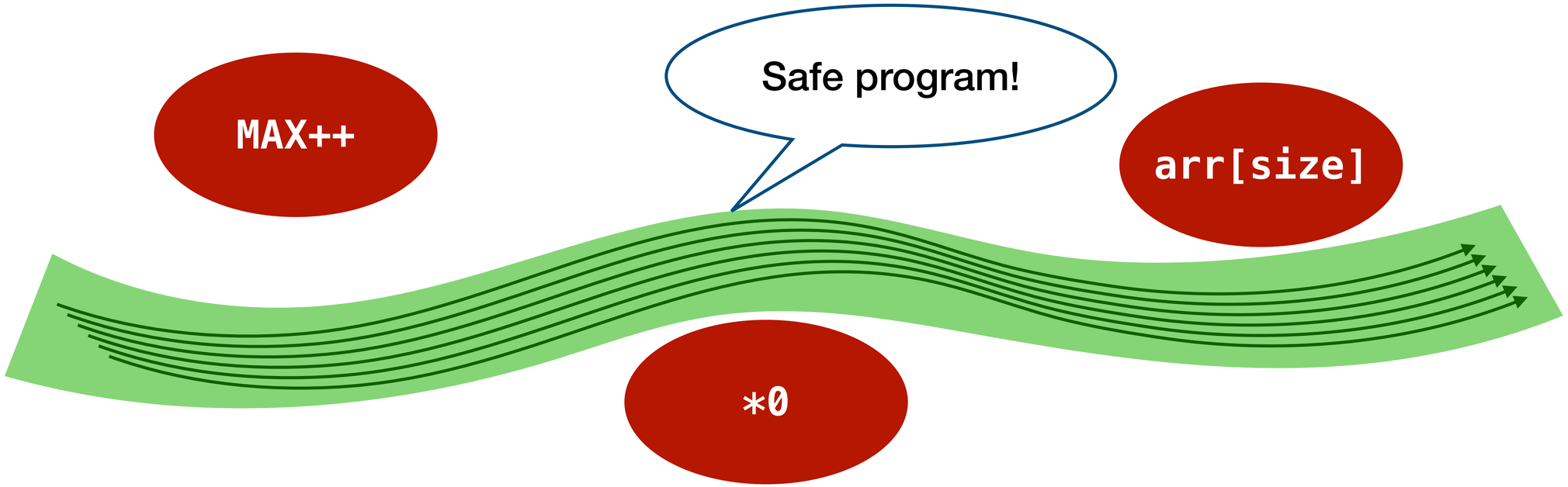
Abstraction of Executions



Abstraction of Executions



Abstraction of Executions



How to analyze?

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

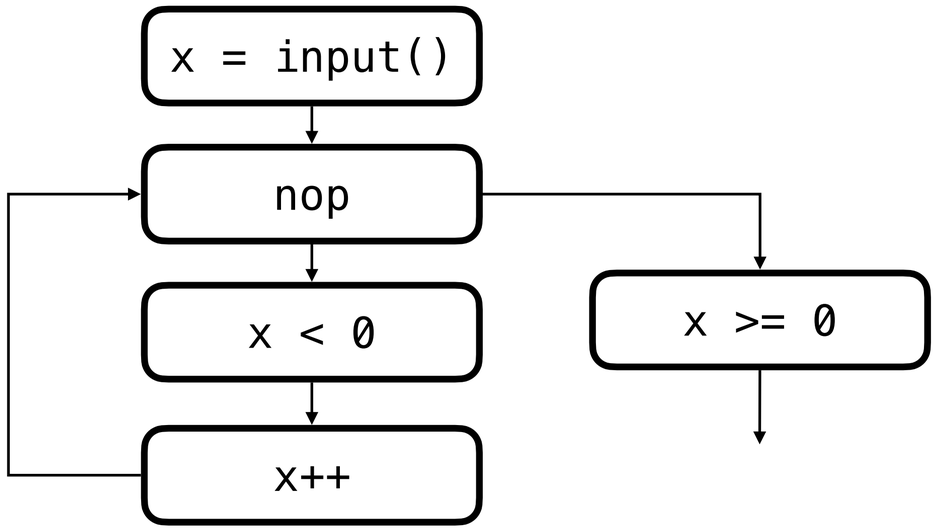
```
x = 3;
while (*) {
  x += 2;
}
x -= 1;
print(x);
```

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 Rules

- 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

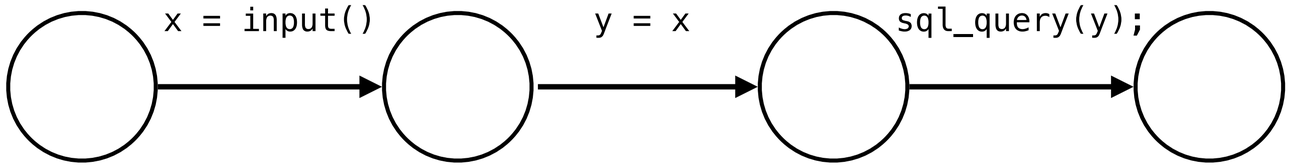
$$\frac{}{\text{true}} \quad \frac{A \qquad B}{A \wedge B} \quad \frac{A \wedge B}{A} \quad \frac{A \wedge B}{B} \quad \frac{A}{A \vee B} \quad \frac{B}{A \vee B} \quad \frac{A \qquad A \implies B}{B} \quad \ldots$$

Closure

- A set is closed under an operation if performing that operation on members of the set always produces a member of that set
- Example: transitive closure of a graph = all reachable nodes from the start nodes

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

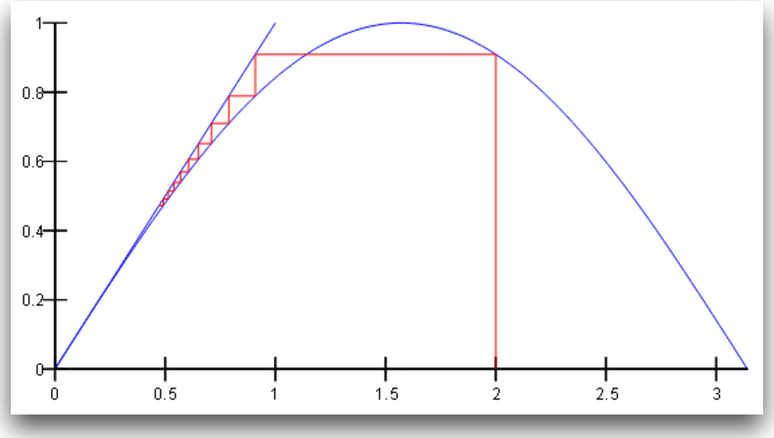
- Computing closure ≈ estimating all possible effects of the operations



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

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);
```

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

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

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

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)**}**

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