IS893: Advanced Software Security

6. Introduction to Static Analysis

Kihong Heo

KAIST

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

```
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

Success Stories

Domain-specific Verification

Windows Device Driver
Microsoft

Airbus Controller
ENS / AbsInt

General-purpose Bug-finding

Stanford / Synopsys

Facebook

SNU / Fasoo.com

Mathworks

GrammaTech

Semmle / Github

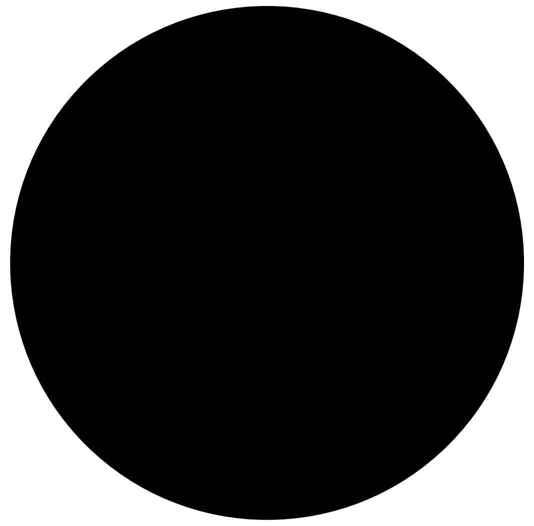
JuliaSoft

GCC

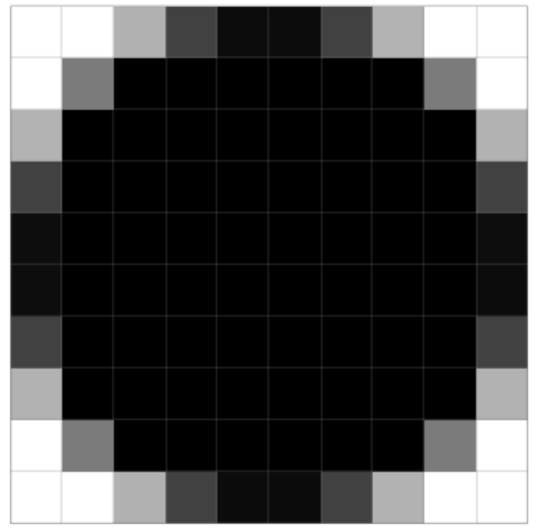
LLVM/Clang

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

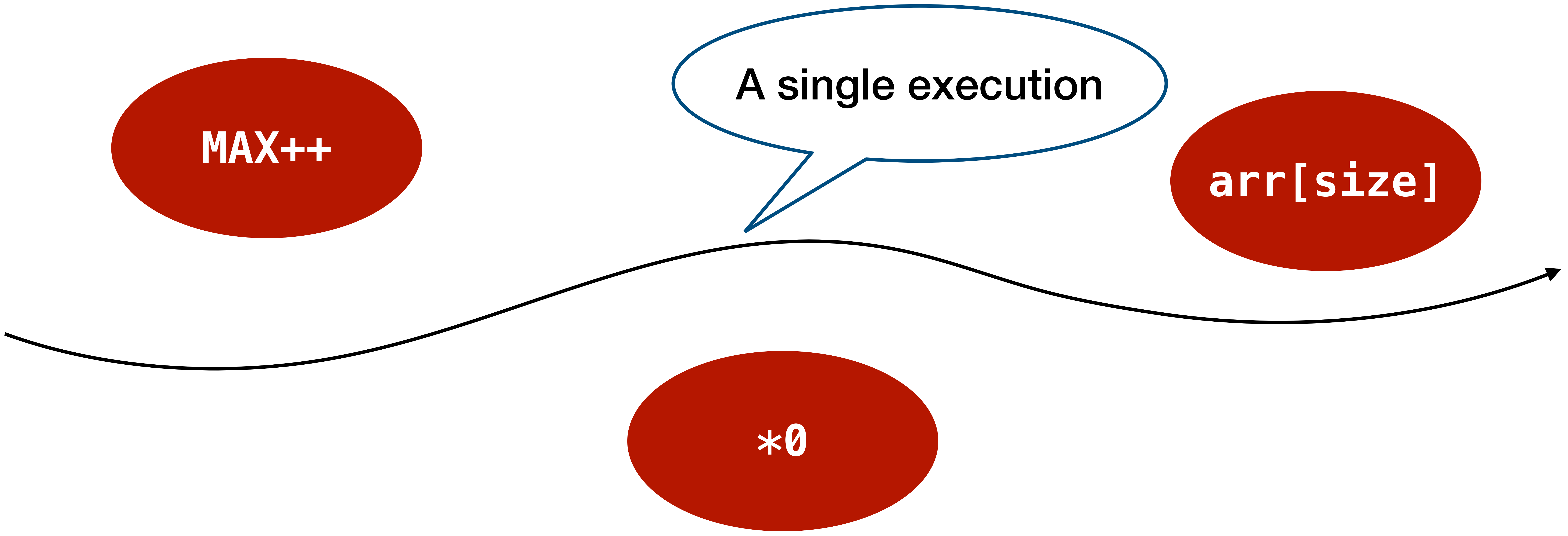
Example

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

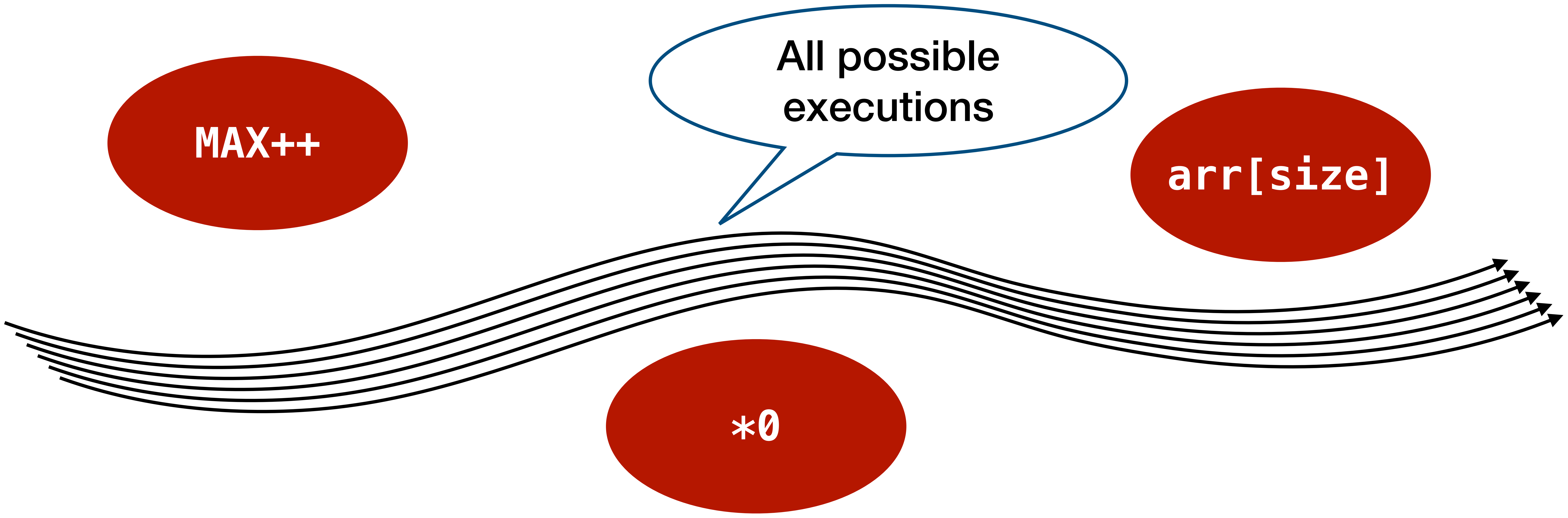
Q: What are the possible output values?

- 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)

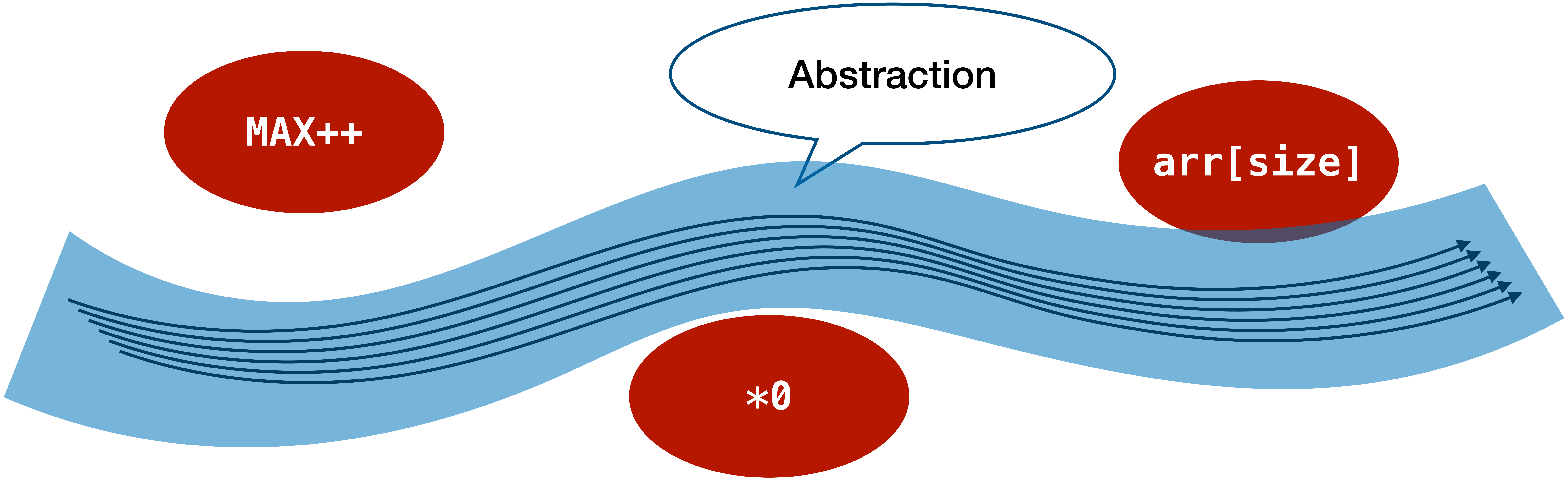
Abstraction of Executions



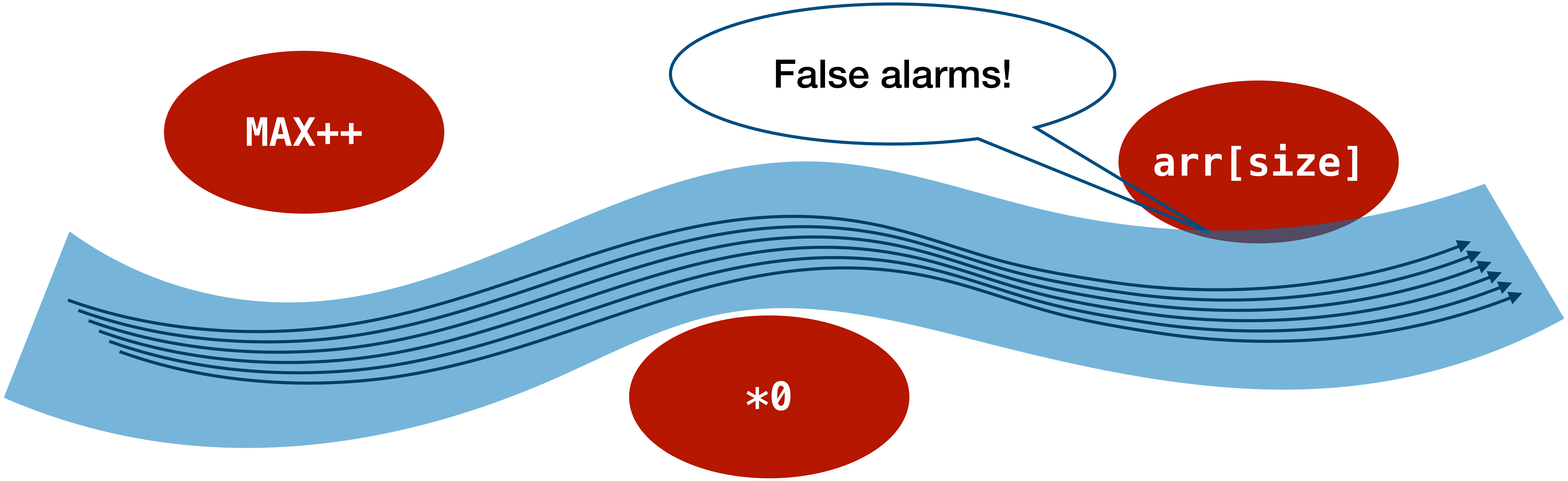
Abstraction of Executions



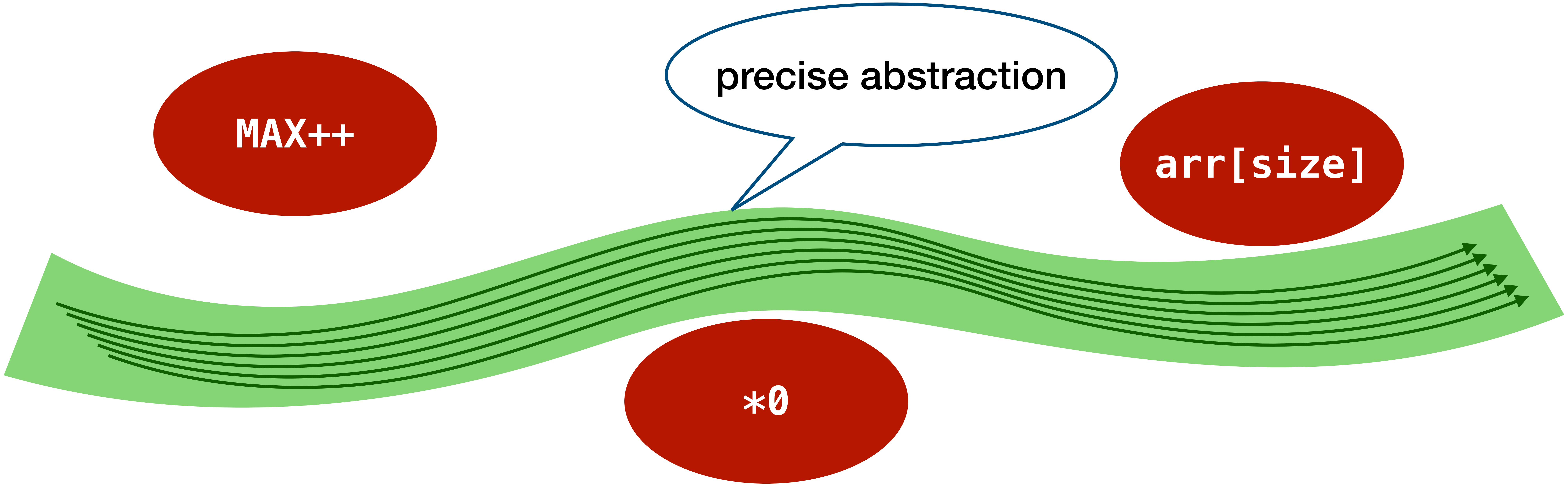
Abstraction of Executions



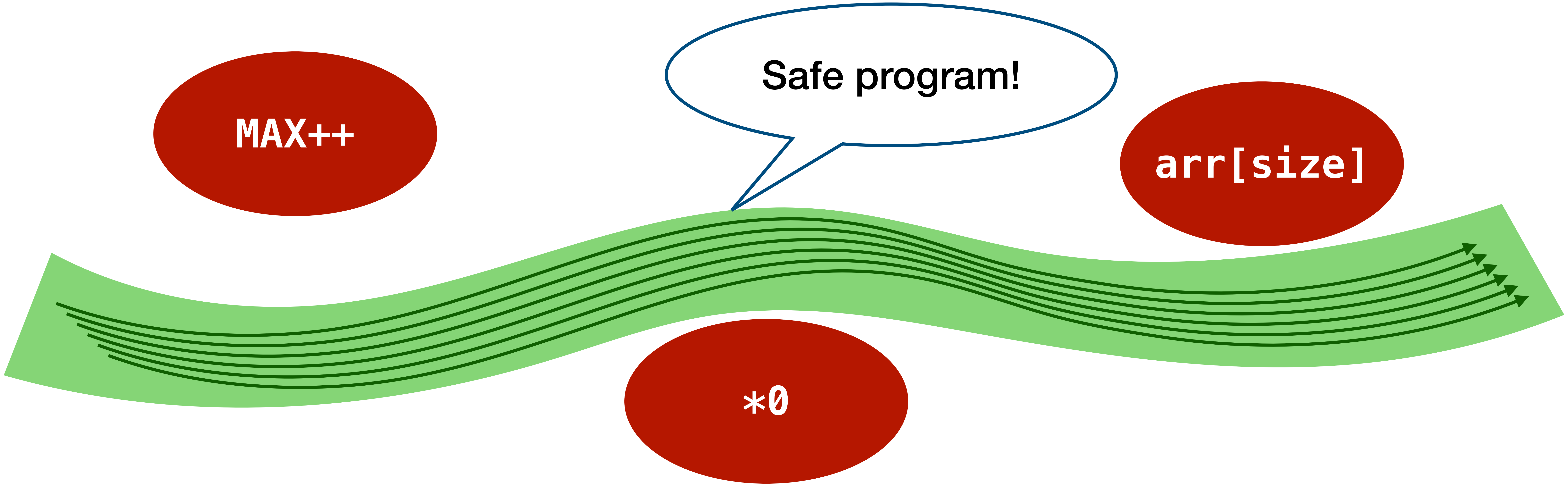
Abstraction of Executions



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

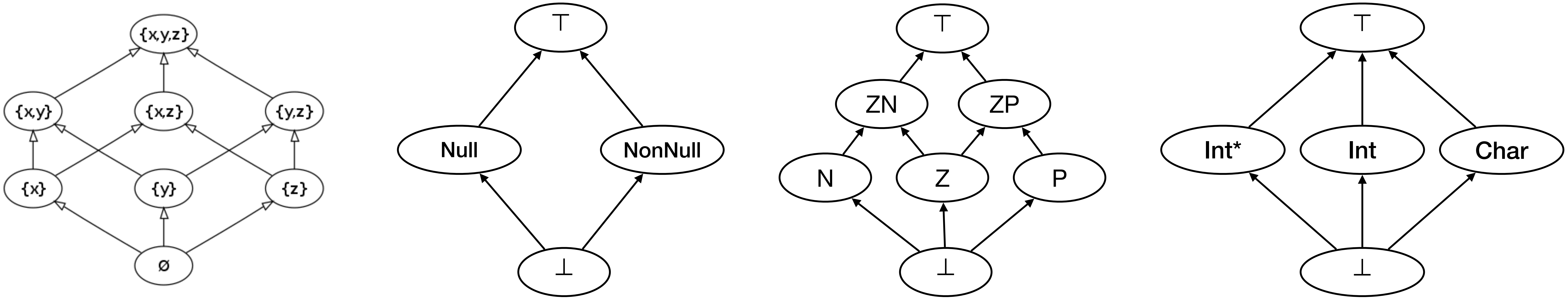
Program	Concrete	Abstract 1	Abstract 2	Abstract 3
`x = 3;`	{3}	Int	Pos	PosOdd
`while (*) {`				
`  x += 2;`				
`}`	{3, 5, 7, …}	Int	Pos	PosOdd
`x -= 1;`	{2, 4, 6, …}	Int	ZeroOrPos	PosEven
`print(x);`				

Abstract Domain

- A set of abstract properties: a crucial design choice for static analysis
- (Semantic) Property: points of interest in programs
    - E.g., “p == NULL?”, “idx < size?”, “fp can be only f, g, or h?”, etc
- Abstract property: approximated semantic properties
    - E.g., Int, Pos, [0,10], NonNull, {f,g,h}, DontKnow, etc

Design of Abstract Domain

- Abstract domain: CPO (complete partial order)
  - “larger” elements have “more” (“imprecise”) information
  - C.f, domain theory
- Example:



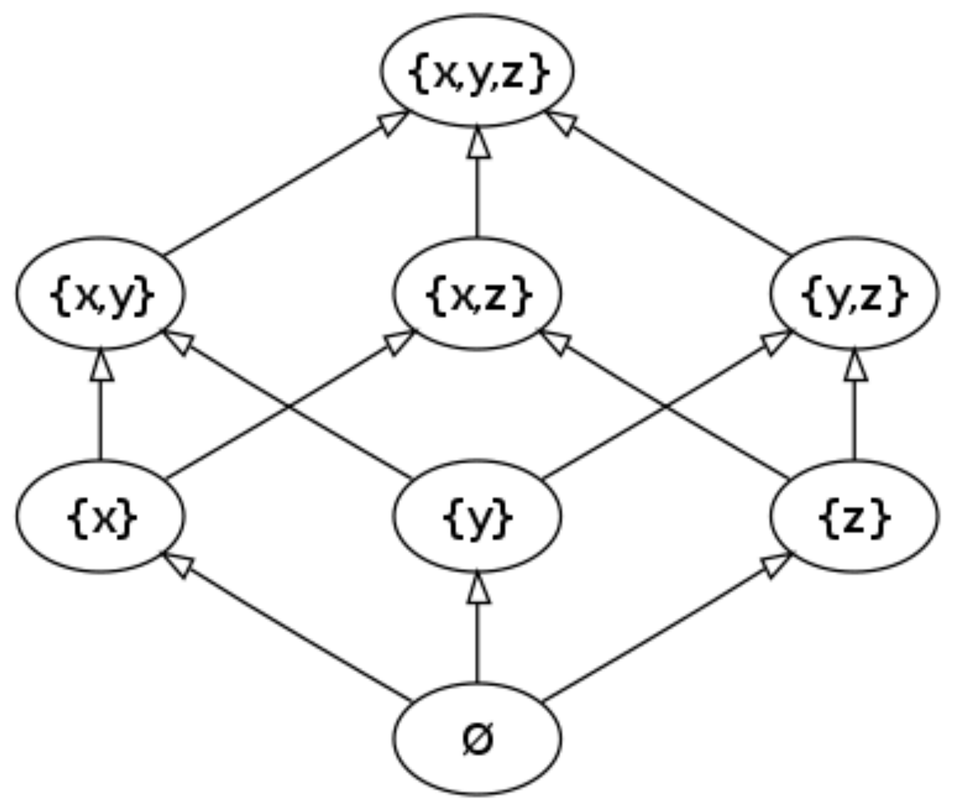
Partial Order

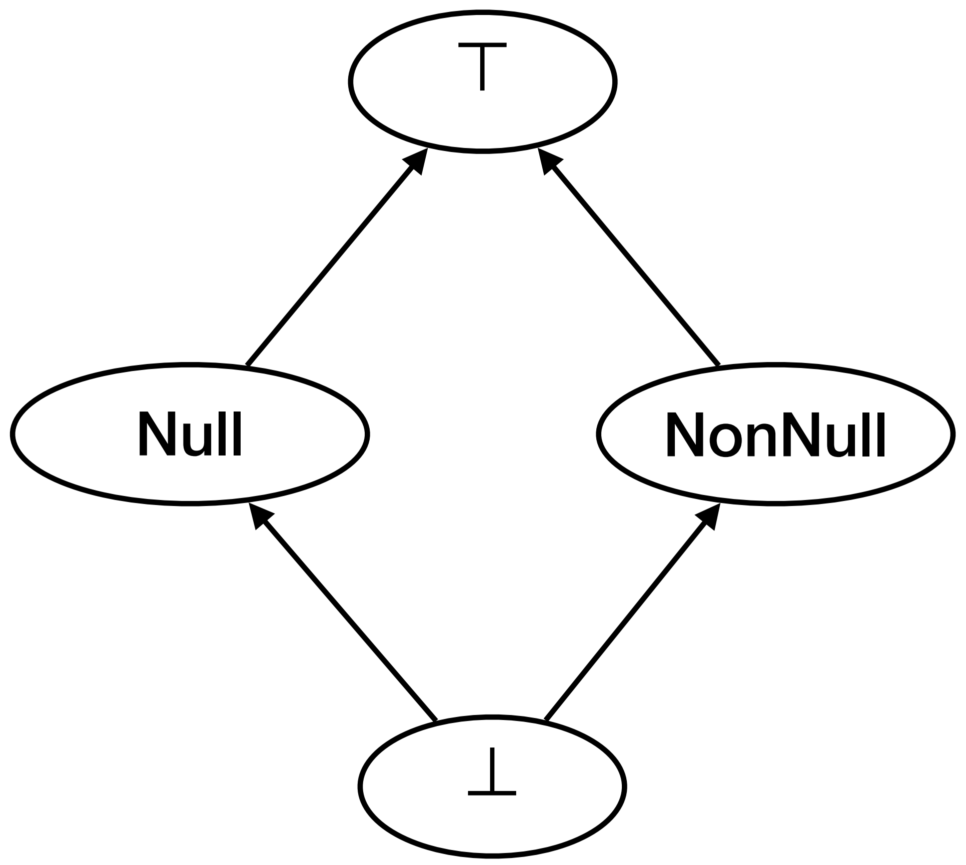
Definition (Partial Order). A binary relation $\sqsubseteq$ is a **partial order** on a set D if it has:

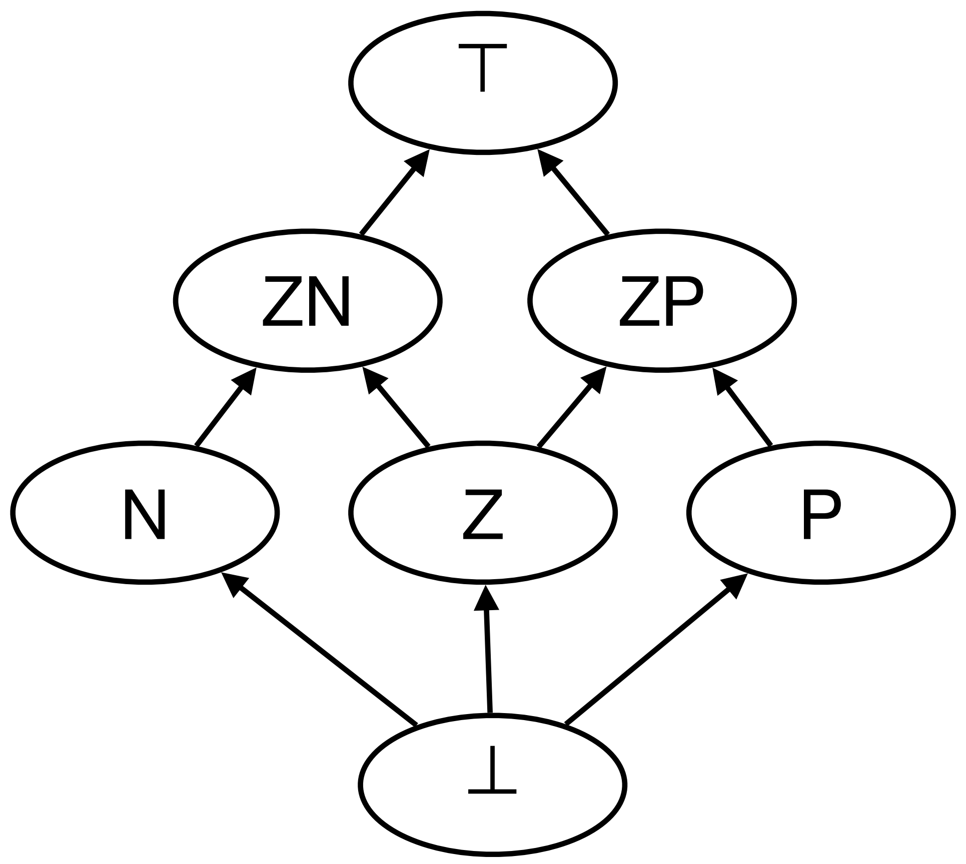
1. reflexivity: $a \sqsubseteq a$ for all $a \in D$
2. Antisymmetry: $a \sqsubseteq b$ and $b \sqsubseteq a$ implies $a = b$
3. Transitivity: $a \sqsubseteq b$ and $b \sqsubseteq c$ implies $a \sqsubseteq c$

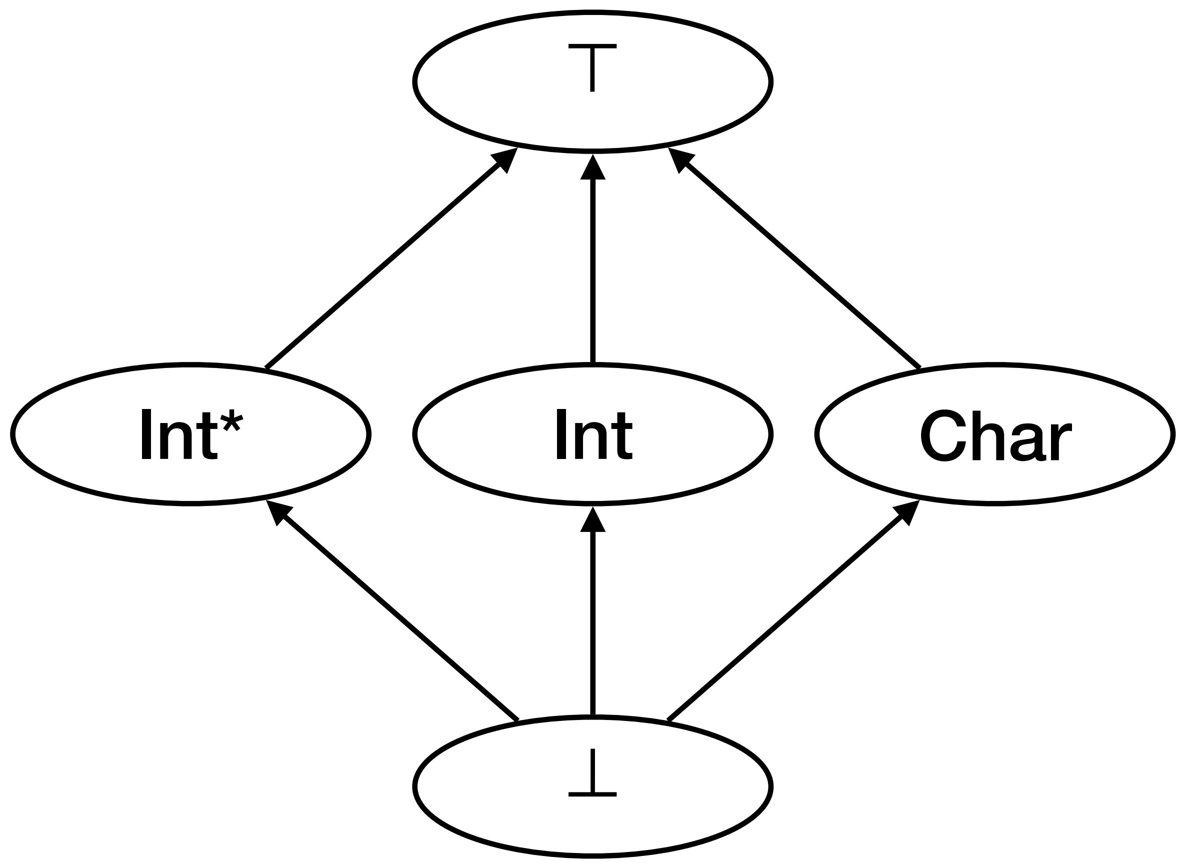
A set D with a partial order $\sqsubseteq$ is called a **partially ordered set** $(D, \sqsubseteq)$, or simply **poset**.

Example









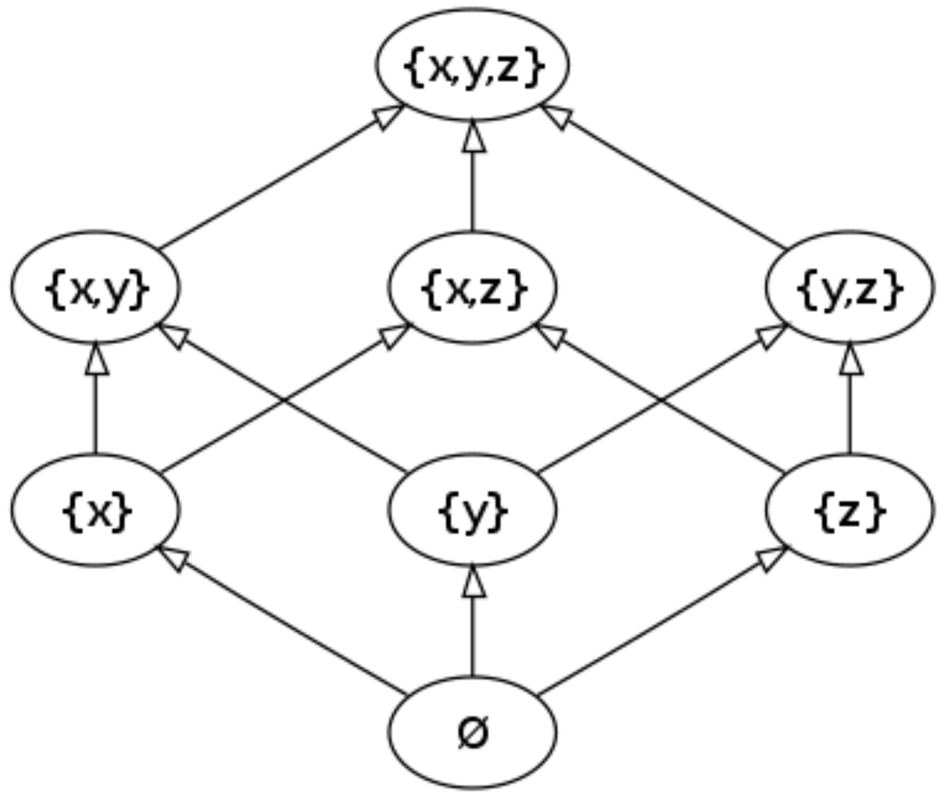
Least Upper Bound (Join)

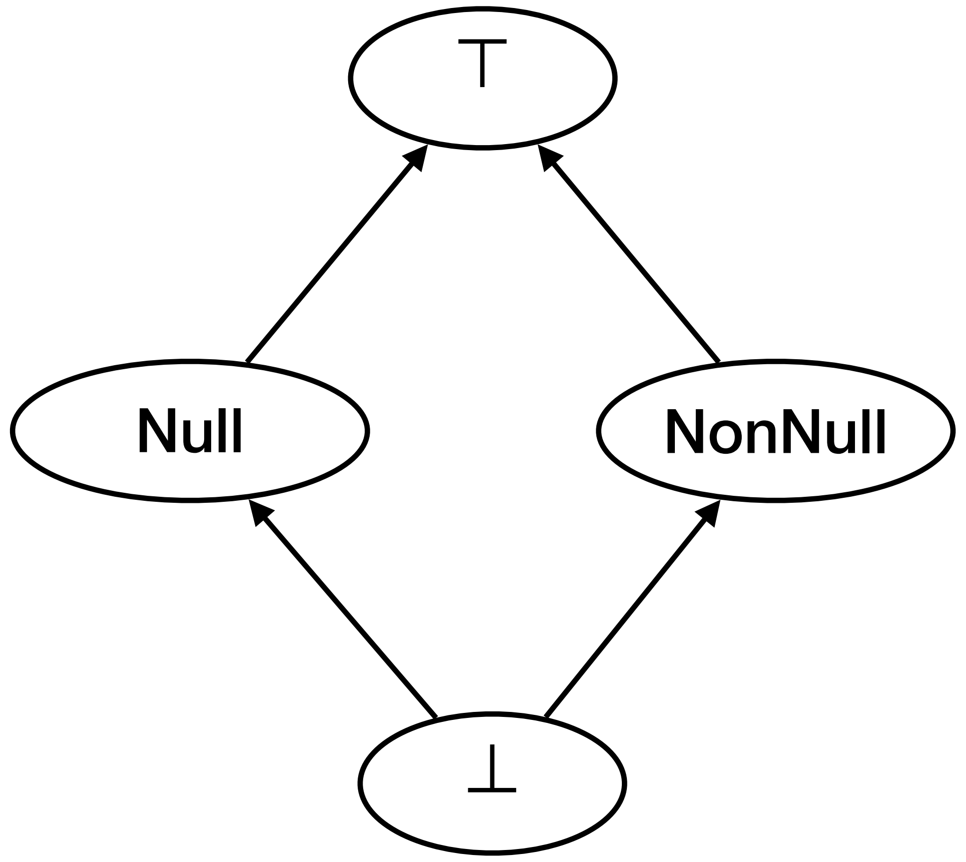
Definition (Least Upper Bound). For a partial ordered set $(D, \sqsubseteq)$ and subset $X \subseteq D$, $d \in X$ is an **upper bound** of X iff

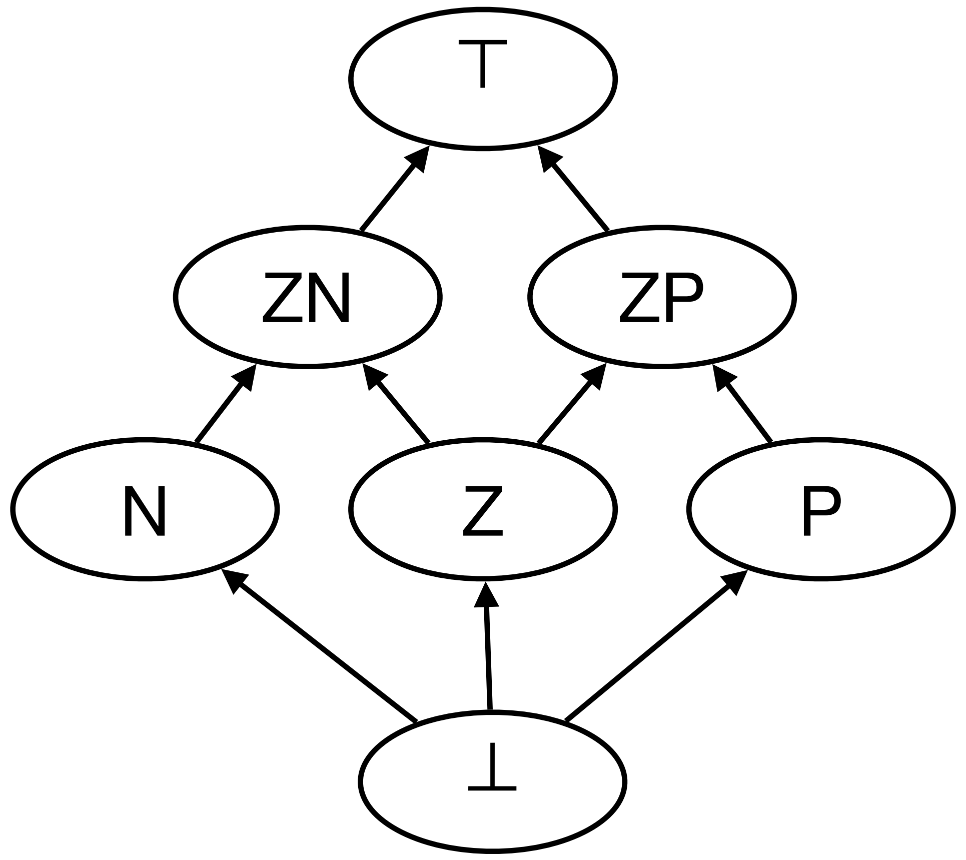
$$\forall x \in X . x \sqsubseteq d.$$

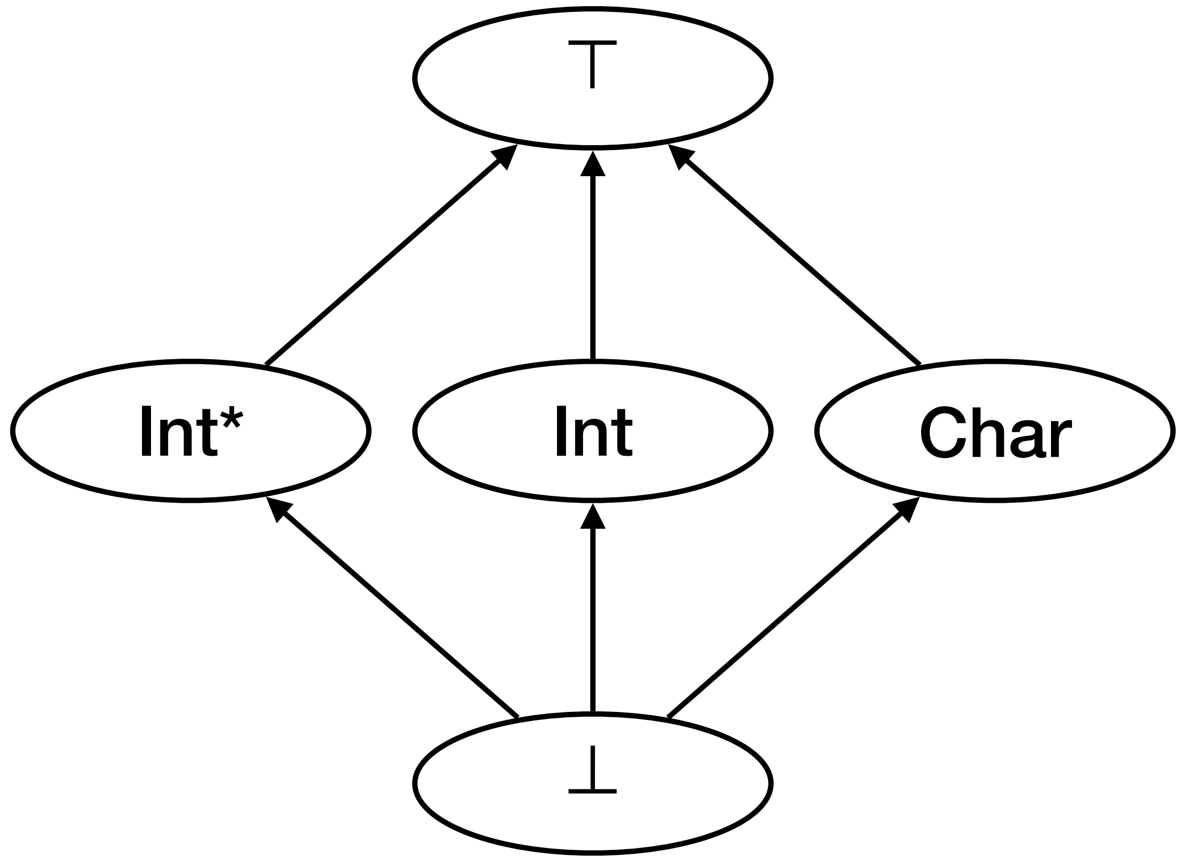
An upper bound d is the **least upper bound** of X iff for all upper bounds y of X, $d \sqsubseteq y$. The least upper bound of X is denoted by $\bigsqcup X$.

Example







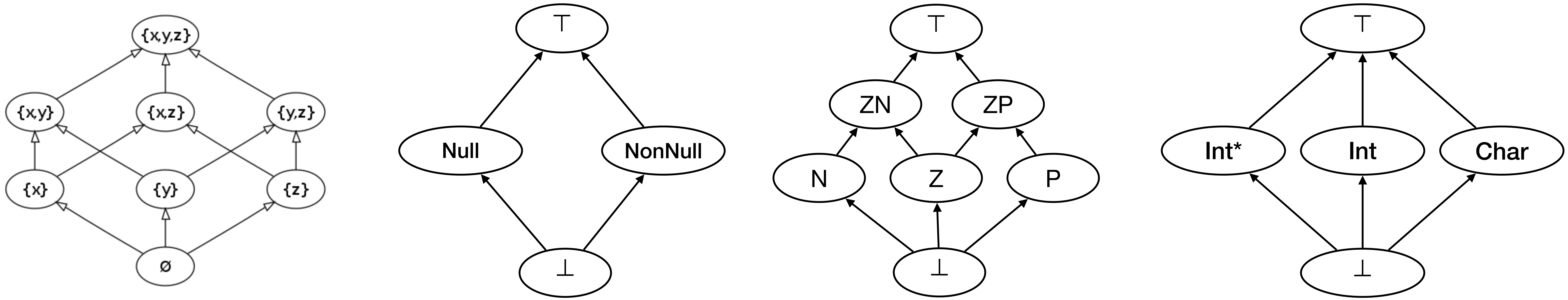


Chain

Definition (Chain). Let $(D, \sqsubseteq)$ be a partial ordered set. A subset $X \subseteq D$ is called **chain** if X is totally ordered:

$$\forall x_1, x_2 \in X \,.\, x_1 \sqsubseteq x_2 \text{ or } x_2 \sqsubseteq x_1 \,.$$

Example

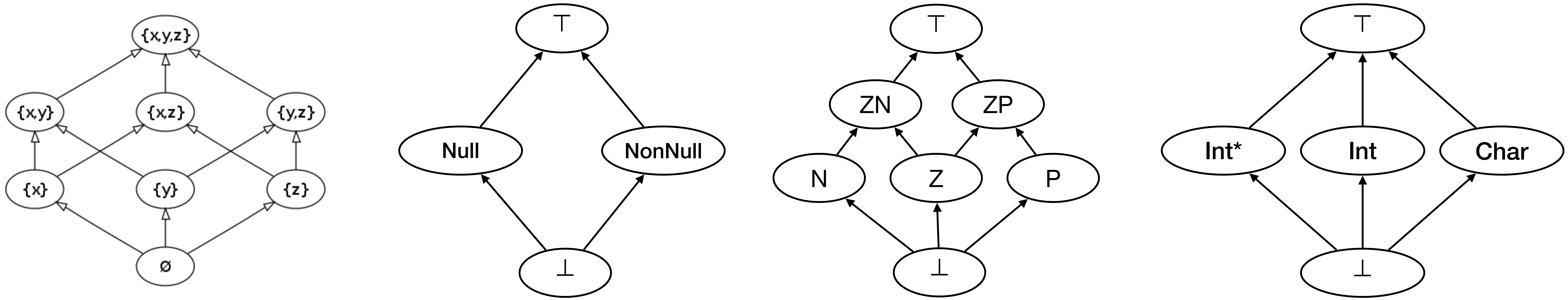


CPO

Definition (CPO). A poset $(D, \sqsubseteq)$ is a **CPO** (complete partial order) if every chain X of D has $\bigsqcup X \in D$.

Lemma. If poset $(D, \sqsubseteq)$ is a CPO, it has the **least element** $\bot = \bigsqcup \emptyset$

Example



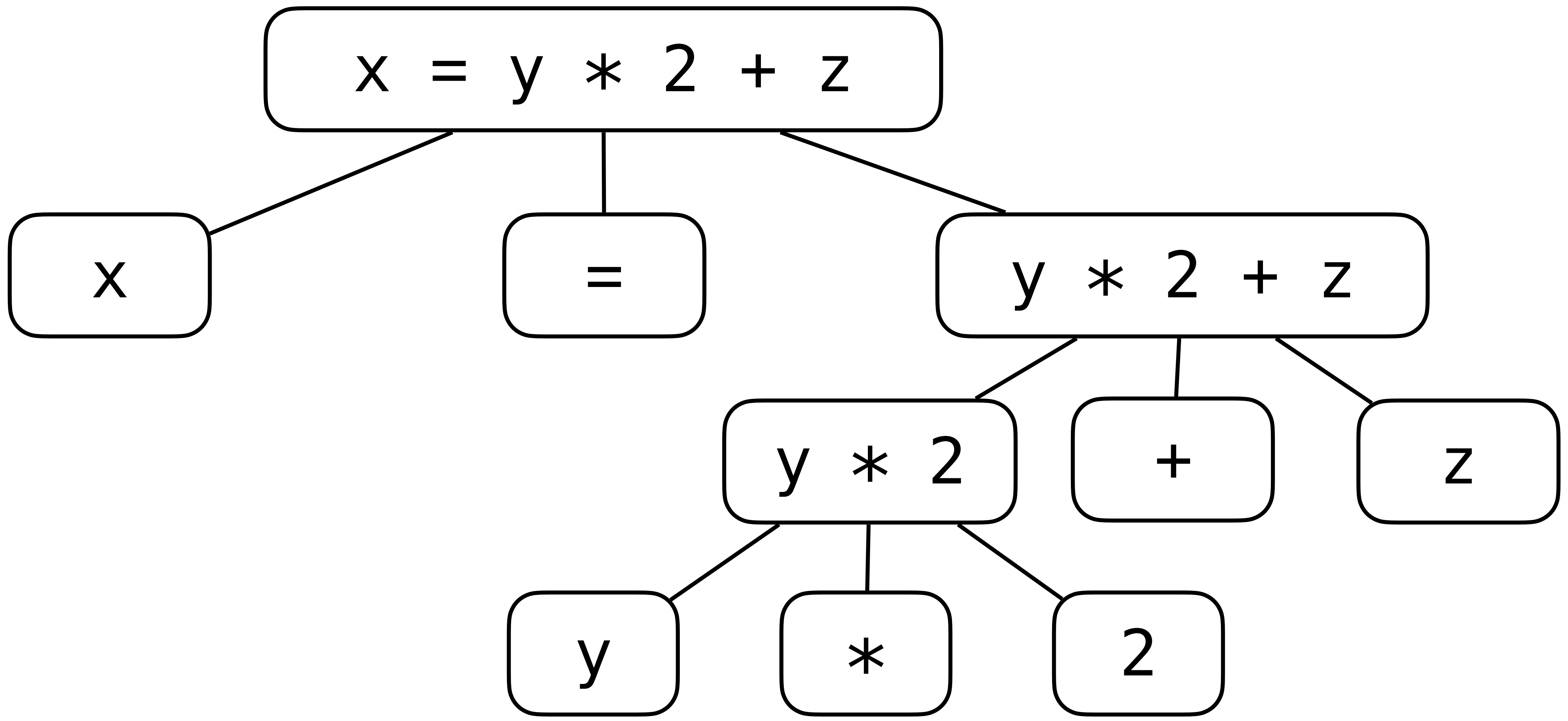
Abstraction & Concretization

- Concretization of an abstract element: estimation of concrete elements
  - Concretization of bottom: non-termination, unreachable, or error
  - Concretization of top: “Unknown”
  - Concretization of X: all possibilities subsumed by X if reachable
- Example:

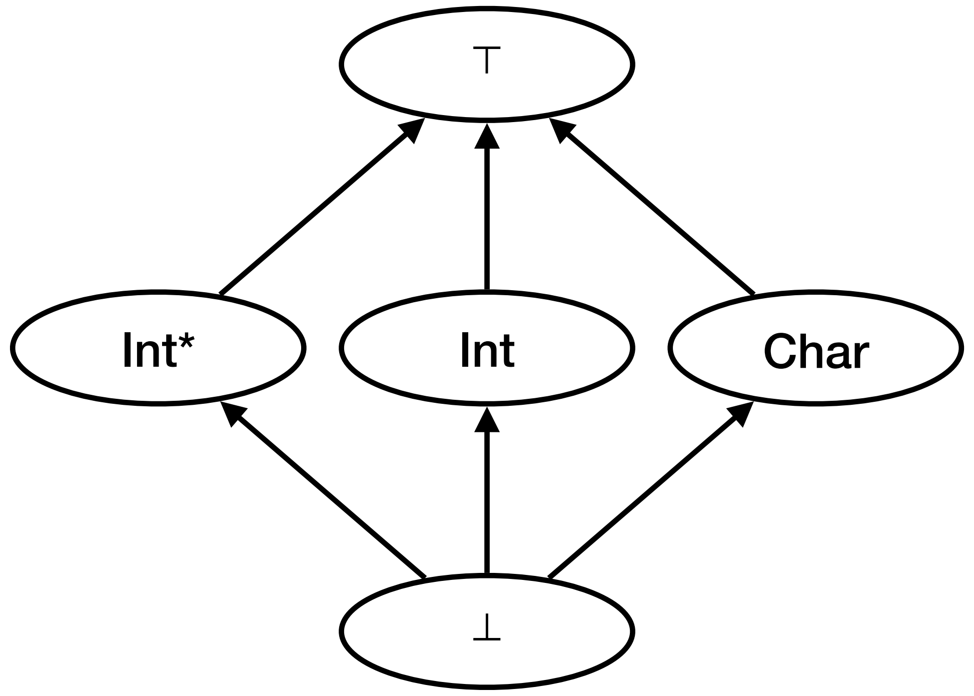
```
x = input();  // Top: any integer (if reachable)
y = 1 / x;    // Top: any integer (if reachable)
z = 1;        // Pos: a positive integer (if reachable)
if (x > 0 && x < 0) {
    // Bot: unreachable
}
```

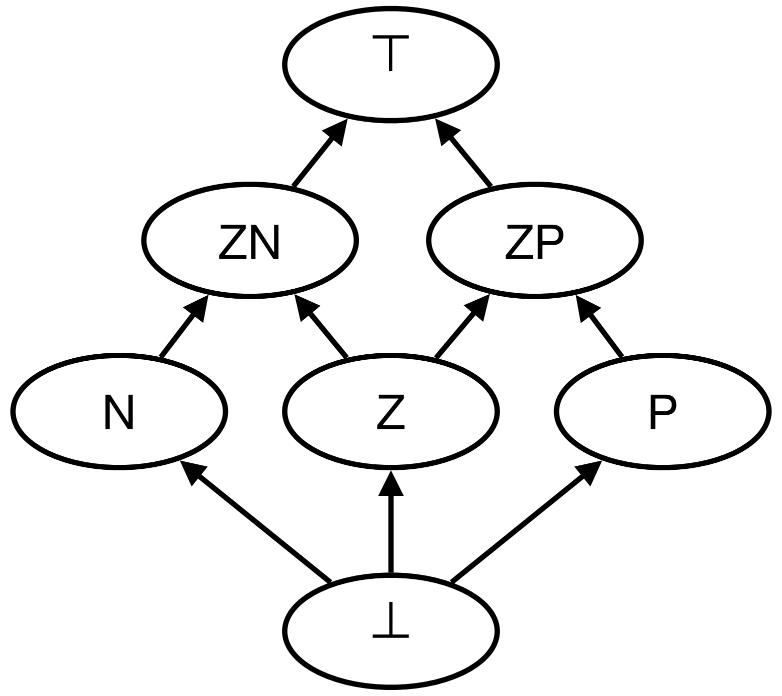
Abstract Semantics

- Abstraction of actual program behavior (i.e., concrete semantics)
- Defined with respect to the underlying abstract domains
- Usually, defined compositionally
  - E.g., abstract semantics of "`x = y * 2 + z`"



Example






```
x = 1;        // x: int
y = 2;        // y: int
if(*)
  r = x * y; // r: int
else
  r = x - y; // r: int
             // r: int
```

```
x = 1;        // x: P
y = 2;        // y: P
if(*)
  r = x * y; // r: P
else
  r = x - 1; // r: ZP
             // r: ZP
```

Design of Static Analysis

- How to design a correct static analysis?
  - General principle & framework: abstract interpretation
  - Specialized frameworks (limited but powerful enough)
    - Static analysis by equations (e.g., data-flow analysis)
    - Static analysis by monotonic closure (e.g., constraint-based analysis)
    - Static analysis by proof construction (e.g., type system)

Conclusion

- Static analysis: a method to estimate SW behavior in advance
- Key: abstraction (approximation)
  - Abstract domain: CPO
  - Abstract semantics: operations on the underlying abstract domains
- Pros: can subsume all possible behavior of SW
- Cons: may have spurious results