# Static & dynamic analysis

CSE 503

# Selecting an abstract domain

$\langle\ x = 2; y = 5\ \rangle$

```
  y = x++;
```

$\langle\ x = 3; y = 2\ \rangle$

$\langle\ x = \{ 3, 5, 7 \}; y = \{ 9, 11, 13 \}\ \rangle$

```
  y = x++;
```

$\langle\ x = \{ 4, 6, 8 \}; y = \{ 3, 5, 7 \}\ \rangle$

$\langle$ x is odd; y is odd $\rangle$

```
  y = x++;
```

$\langle$ x is even; y is odd $\rangle$

$\langle x{=}3, y{=}11\rangle, \langle x{=}5, y{=}9\rangle, \langle x{=}7, y{=}13\rangle$

```
  y = x++;
```

$\langle x{=}4, y{=}3\rangle, \langle x{=}6, y{=}5\rangle, \langle x{=}8, y{=}7\rangle$

$\langle$ x is prime; y is prime $\rangle$

```
  y = x++;
```

$\langle$ x is anything; y is prime $\rangle$

$\langle\ x_n = f(a_{n-1},\ldots,z_{n-1}); y_n = f(a_{n-1},\ldots,z_{n-1})\ \rangle$

```
  y = x++;
```

$\langle\ x_{n+1} = x_n + 1; y_{n+1} = x_n\ \rangle$

# Analysis result:  positive and negative

Ideal analysis outputs:   “program is wrong” or “program is right”

# Analysis result:  positive and negative

Ideal analysis outputs:   “program is wrong” or “program is right”

Actual analysis outputs:

- “Program might be wrong” or “program is right”
- “Program is wrong” or “program might be right”

# Analysis result:  positive and negative

Ideal analysis outputs:   “program is wrong” or “program is right”

Actual analysis outputs:

- “Program might be wrong” or “program is right”     verification
- “Program is wrong” or “program might be right”     linting

“Positive” = “alarm” = “program might be wrong”

“Negative” = “OK” = “program is right”

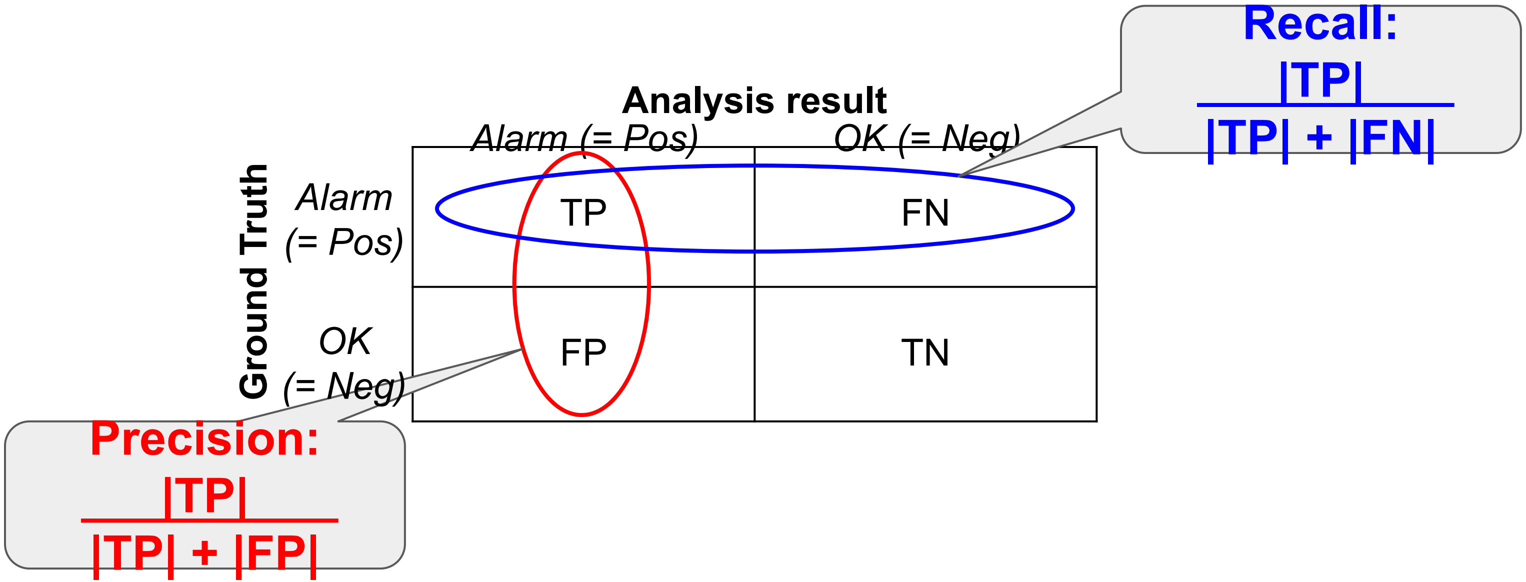
# True/false and positive/negative

| | | **Analysis result** | |
|---|---|---|---|
| | | *Alarm (= Pos)* | *OK (= Neg)* |
| **Ground Truth** | *Alarm (= Pos)* | | |
| | *OK (= Neg)* | | |

# True/false and positive/negative

| | | **Analysis result** | |
|---|---|---|---|
| | | *Alarm (= Pos)* | *OK (= Neg)* |
| **Ground Truth** | *Alarm (= Pos)* | TP | FN |
| | *OK (= Neg)* | FP | TN |

# Precision vs recall (and FP/FN/TP/TN)



| | | Analysis result | |
|---|---|---|---|
| | | Alarm (= Pos) | OK (= Neg) |
| Ground Truth | Alarm (= Pos) | TP | FN |
| | OK (= Neg) | FP | TN |

**Recall:**

$$\frac{|TP|}{|TP| + |FN|}$$

**Precision:**

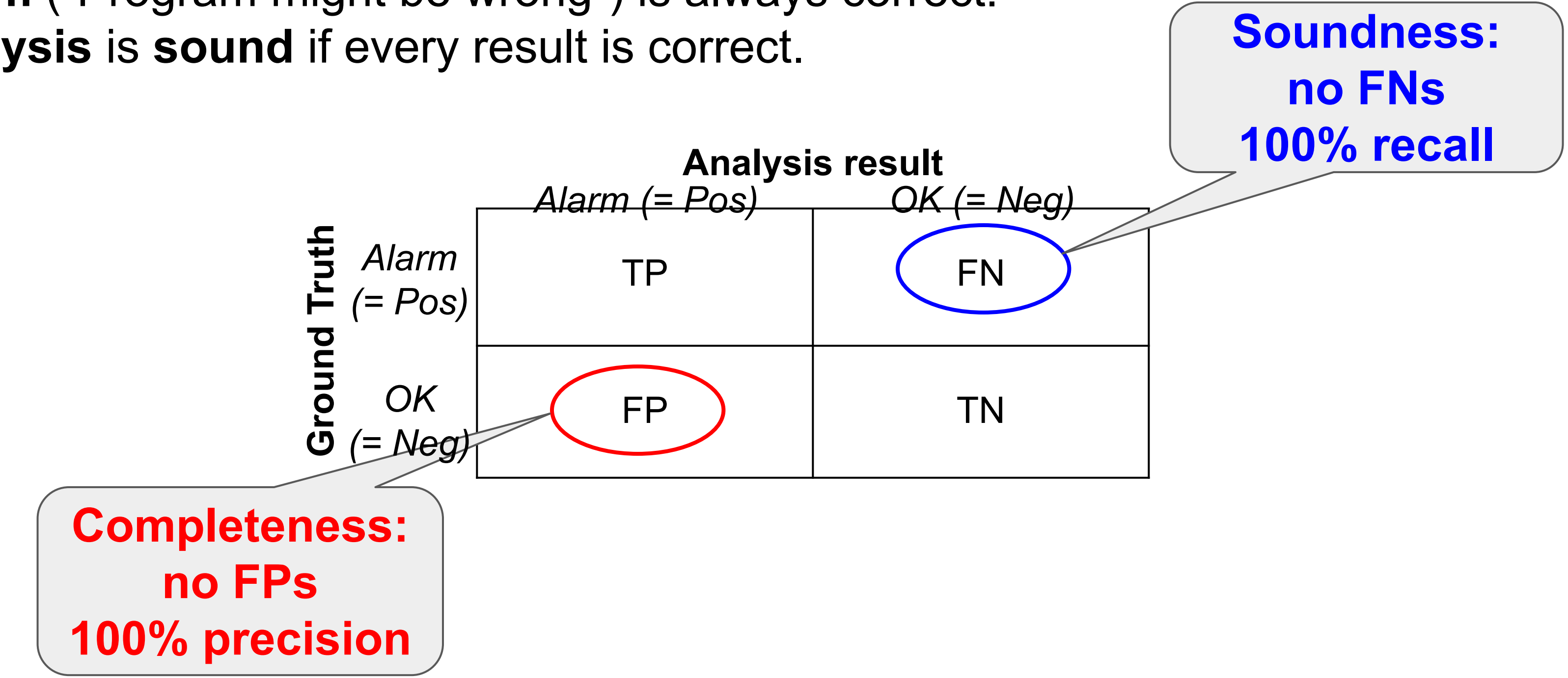
$$\frac{|TP|}{|TP| + |FP|}$$

# Soundness vs. completeness

| | | **Analysis result** | |
|---|---|---|---|
| | | *Alarm (= Pos)* | *OK (= Neg)* |
| **Ground Truth** | *Alarm (= Pos)* | TP | FN |
| | *OK (= Neg)* | FP | TN |

# Soundness vs. completeness

A **result** is correct or incorrect (or is a TP/FP/FN/TN).
An **alarm** (“Program might be wrong”) is always correct.
An **analysis** is **sound** if every result is correct.



|  |  | **Analysis result** |  |
|---|---|---|---|
|  |  | *Alarm (= Pos)* | *OK (= Neg)* |
| **Ground Truth** | *Alarm (= Pos)* | TP | FN |
|  | *OK (= Neg)* | FP | TN |

**Soundness:**
**no FNs**
**100% recall**

**Completeness:**
**no FPs**
**100% precision**

# Concrete domain vs. abstract domain

# Concrete domain vs. abstract domain

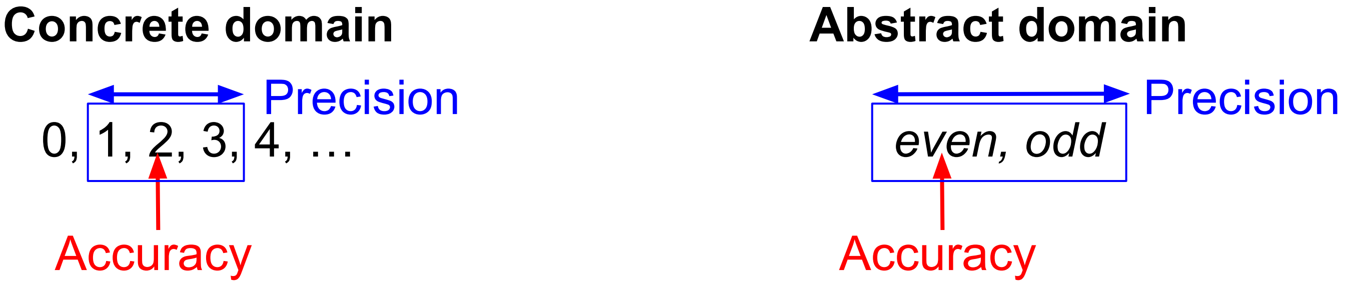
**Concrete domain**

0, 1, 2, 3, 4, …

**Abstract domain**

*even, odd*

# Accuracy vs. precision



Accuracy = correct estimate (guaranteed if sound analysis)

Precision = small estimate

# Any analysis can be done statically or dynamically

- Type safety:  no memory corruption or operations on wrong types of values
    - Static type-checking
    - Dynamic type-checking

- Slicing:  what computations could affect a value
    - Static:  reachability over dependence graph
    - Dynamic:  tracing

# Memory checking

Goal:  find array bound violations, uses of uninit. memory

Purify [Hastings 92], Valgrind:  run-time instrumentation

- Tagged memory:  2 bits (allocated, initialized) per byte
- Each instruction checks/updates the tags
  - Allocate:  set “A” bit, clear “I” bit
  - Write:  require “A” bit, set “I” bit
  - Read:  require “I” bit
  - Deallocate:  clear “A” bit

LCLint [Evans 96]:  compile-time dataflow analysis

- Abstract state contains allocated and initialized bits
- Each transfer function checks/updates the state

Identical analyses!

Another example:  atomicity checking [Flanagan 2003]

# Specifications

- Specification checking
  - Statically: theorem-proving
  - Dynamically: **assert** statement

- Specification generation
  - Statically: by hand or abstract interpretation [Cousot 77]
  - Dynamically: by invariant detection [Ernst 99], reporting unfalsified properties

# More analogous analyses

When you have a problem, consider both static and dynamic approaches