CSE 403: Software Engineering, Spring 2015

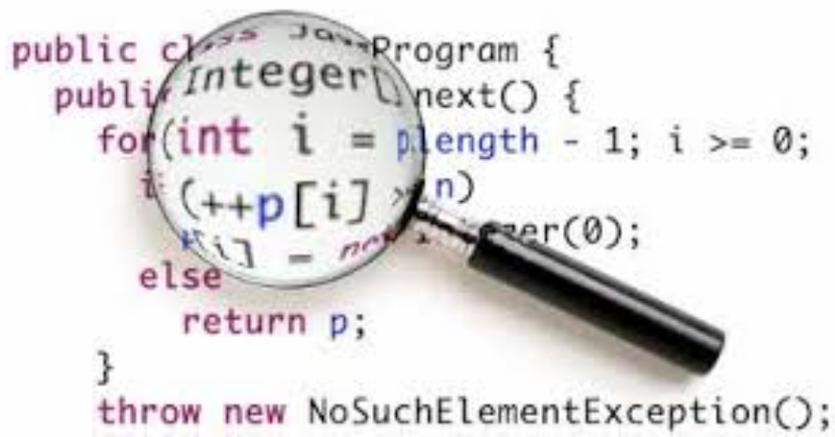
courses.cs.washington.edu/courses/cse403/15sp/

Static Analysis

Emina Torlak
emina@cs.washington.edu

Outline

- What is static analysis?
- How does it work?
- Free and commercial tools



a brief introduction to static analysis

What is static analysis?

- A **static analysis tool** S analyzes the source code of a program P to determine whether it satisfies a property φ, such as
    - "P never deferences a null pointer"
    - "P does not leak file handles"
    - "No cast in P will lead to a ClassCastException"
    - …

What is static analysis?

- A **static analysis tool** S analyzes the source code of a program P to determine whether it satisfies a property φ, such as
  - "P never deferences a null pointer"
  - "P does not leak file handles"
  - "No cast in P will lead to a ClassCastException"
  - …

But it is impossible to write such a tool! For any nontrivial property φ, there is no general automated method to determine whether P satisfies φ (Rice's theorem).

What is static analysis?

- A **static analysis tool** S analyzes the source code of a program P to determine whether it satisfies a property φ, such as
  - "P never deferences a null pointer"
  - "P does not leak file handles"
  - "No cast in P will lead to a ClassCastException"
  - …

But it is impossible to write such a tool! For any nontrivial property φ, there is no general automated method to determine whether P satisfies φ (Rice's theorem).

So, why are we having this lecture?

What is practical static analysis?

What is practical static analysis?

- A **static analysis tool** S analyzes the source code of a program P to determine whether it satisfies a property φ, but it can be wrong in one of two ways:

What is practical static analysis?

- A **static analysis tool** S analyzes the source code of a program P to determine whether it satisfies a property φ, but it can be wrong in one of two ways:
    - If S is **sound**, it will never miss any violations, but it may say that P violates φ even though it doesn't (resulting in **false positives**).

What is practical static analysis?

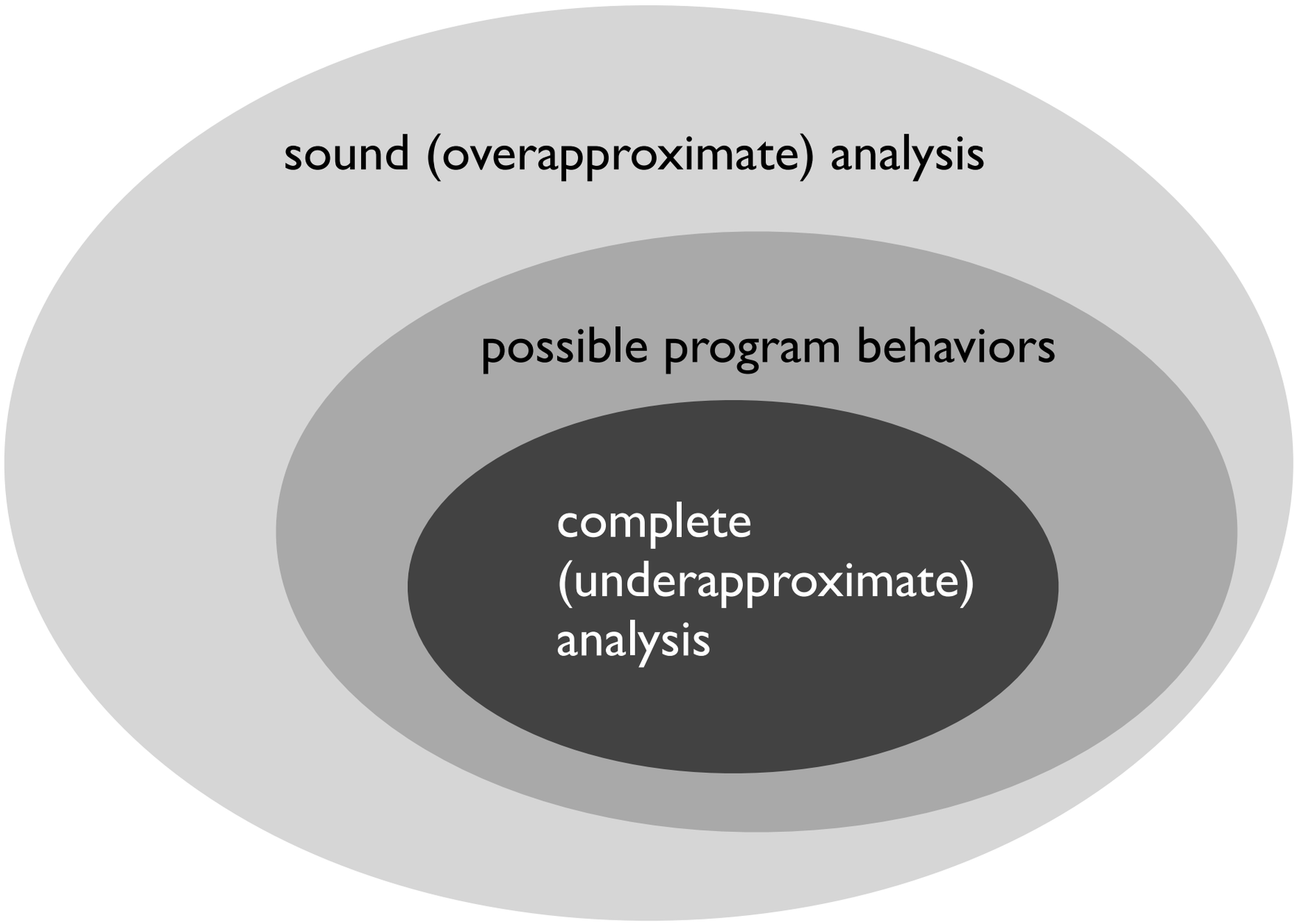
- A **static analysis tool** S analyzes the source code of a program P to determine whether it satisfies a property φ, but it can be wrong in one of two ways:
    - If S is **sound**, it will never miss any violations, but it may say that P violates φ even though it doesn't (resulting in **false positives**).
    - If S is **complete**, it will never report false positives, but it may miss real violations of φ (resulting in **false negatives**).

What is practical static analysis?

- A **static analysis tool** S analyzes the source code of a program P to determine whether it satisfies a property φ, but it can be wrong in one of two ways:
    - If S is **sound**, it will never miss any violations, but it may say that P violates φ even though it doesn't (resulting in **false positives**).
    - If S is **complete**, it will never report false positives, but it may miss real violations of φ (resulting in **false negatives**).

What is a trivial way to implement a sound analysis? A complete analysis?

Soundness vs completeness



Applications of static analysis

- Compilers (sound)
    - type checking, liveness analysis, alias analysis, …
- Bug finding (usually complete)
- Verification (sound)

static analysis by example

A toy static analysis: find a computation's sign

A toy static analysis: find a computation's sign

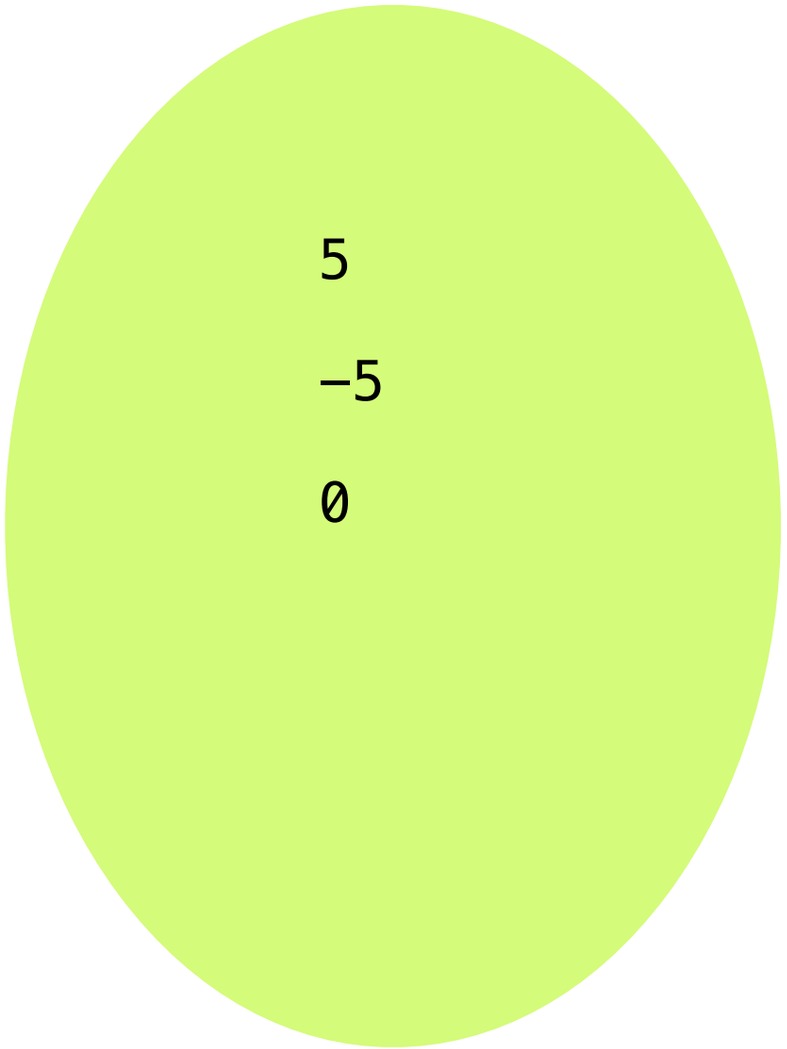
- Given a program P, determine the sign (positive, negative, or zero) of all of its variables.

A toy static analysis: find a computation's sign

- Given a program P, determine the sign (positive, negative, or zero) of all of its variables.
- Applications:
    - Check for division by 0
    - Optimize by storing + variables as unsigned integers
    - Check for negative array indices
    - …

A toy static analysis: abstraction

concrete domain of ints



abstract domain of signs

⊕ positive ints

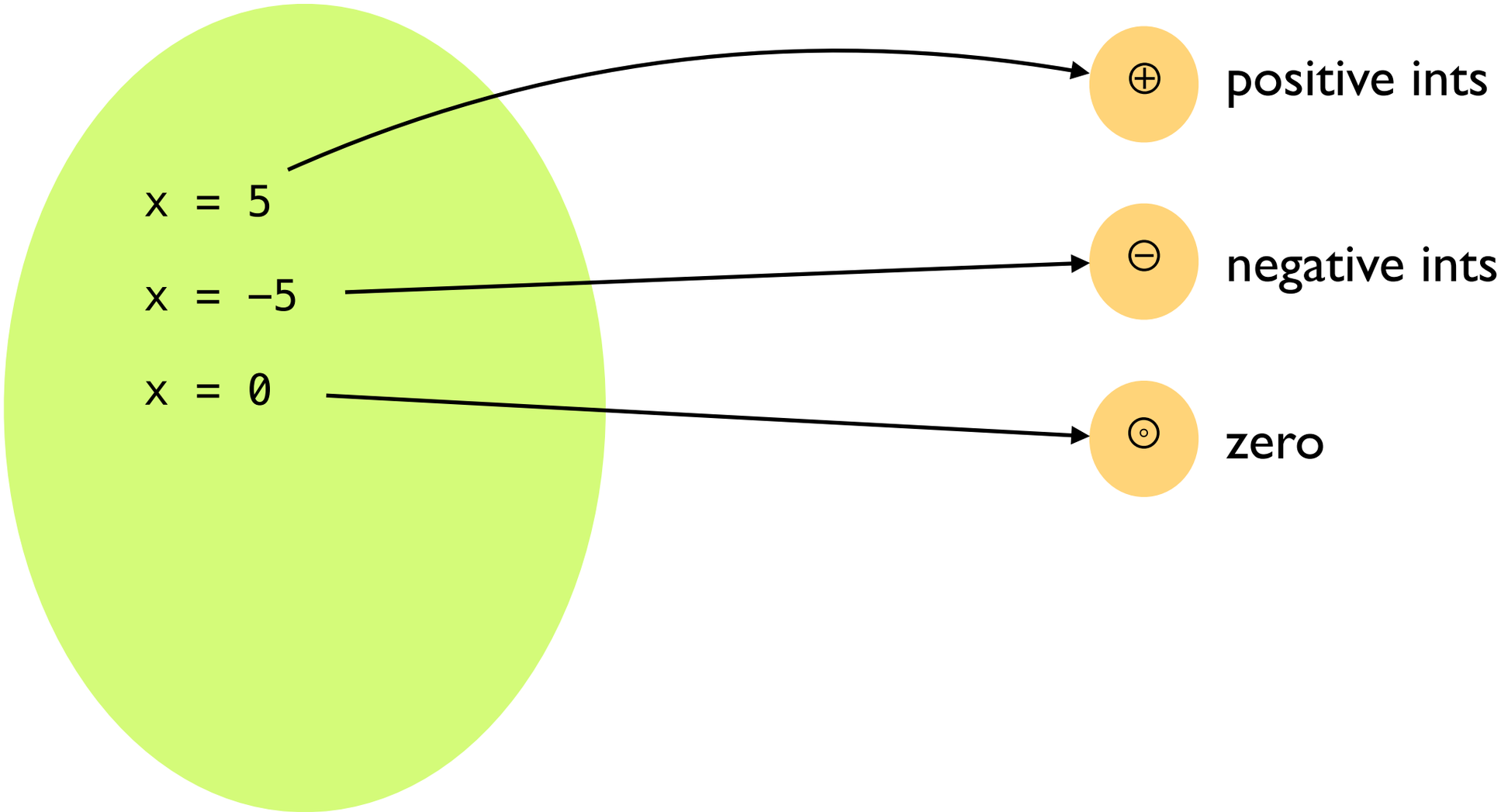
⊖ negative ints

⊙ zero

A toy static analysis: abstraction

concrete domain of ints

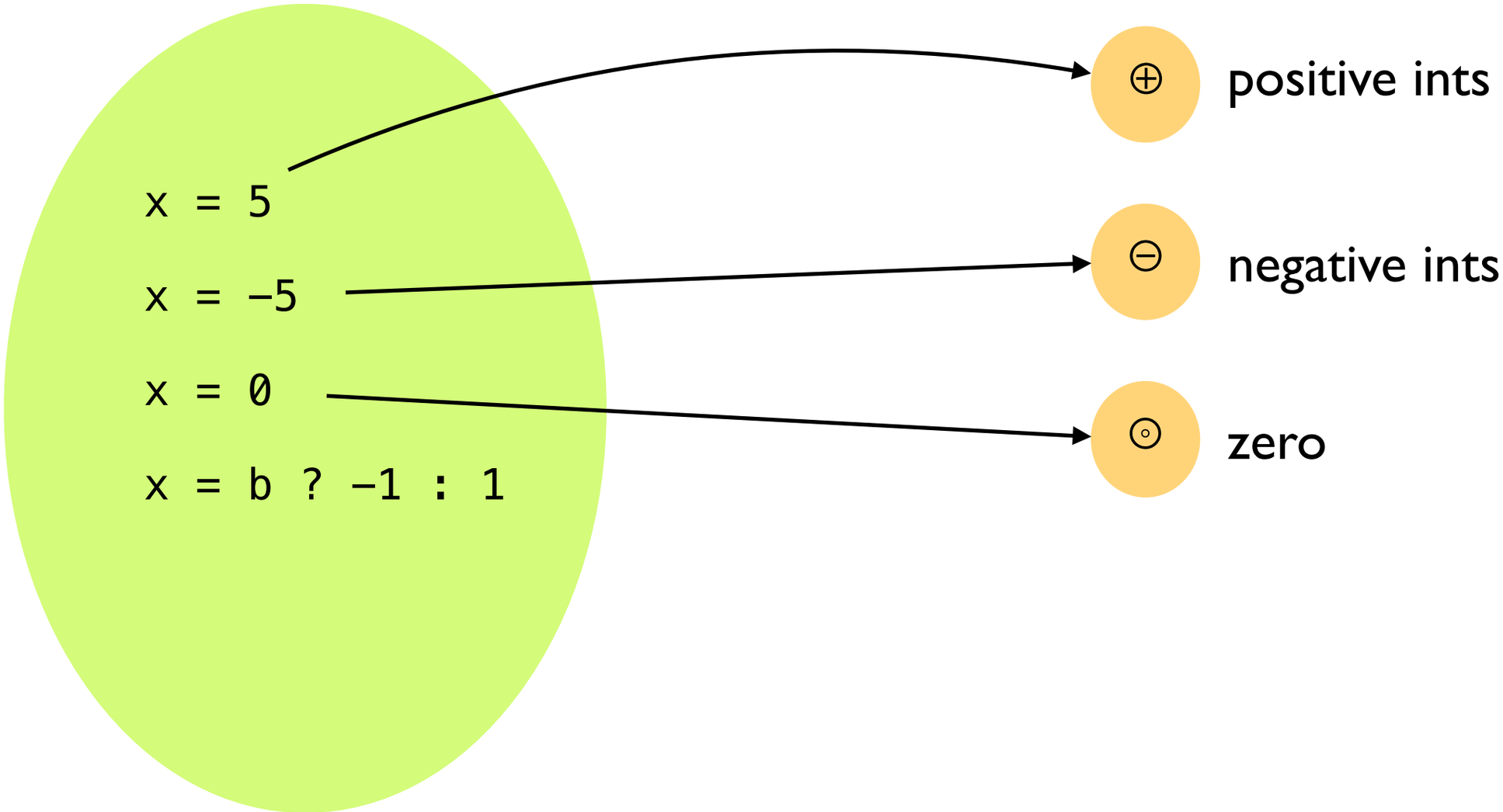
abstract domain of signs



A toy static analysis: abstraction

concrete domain of ints

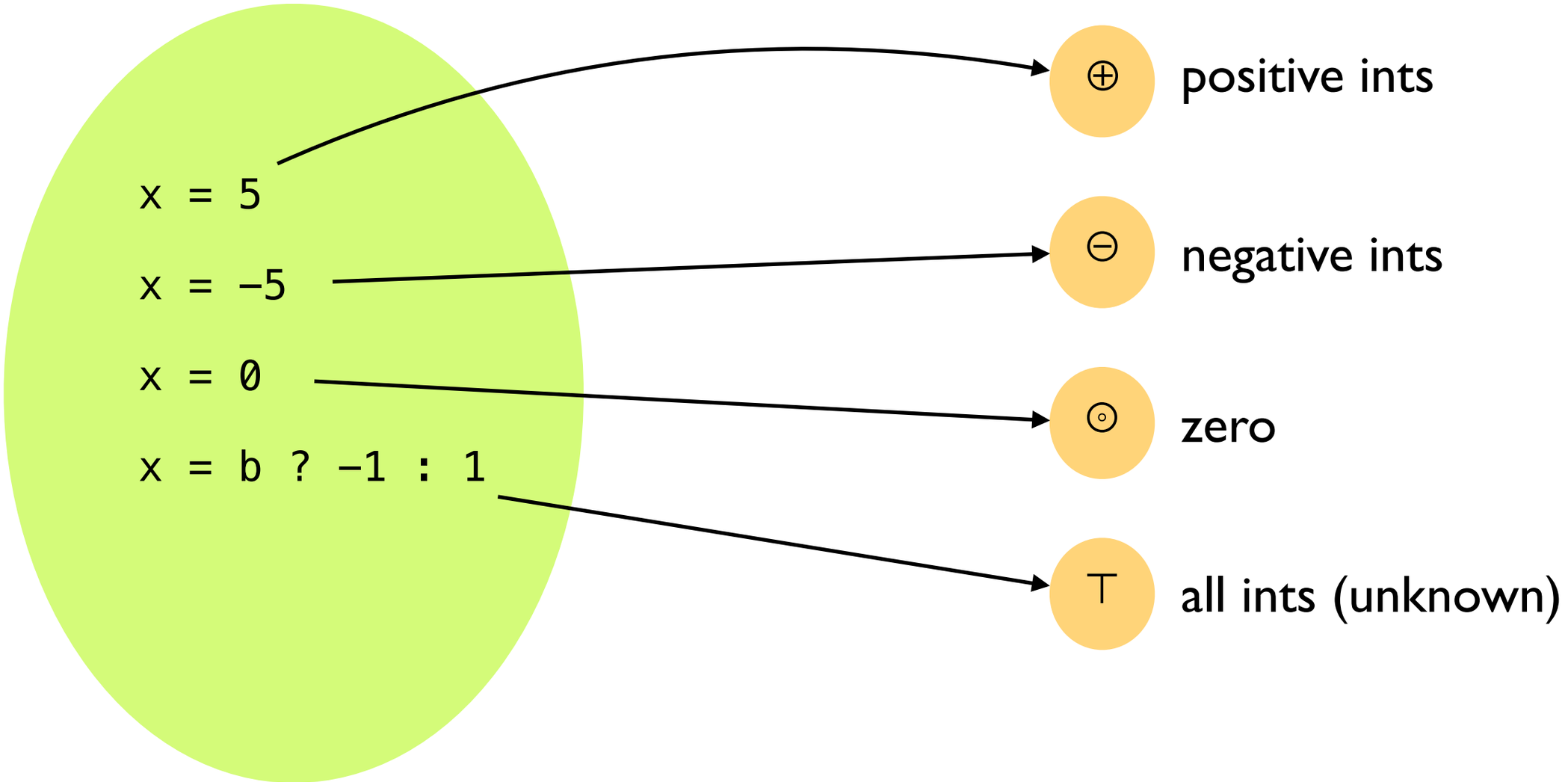
abstract domain of signs



A toy static analysis: abstraction

concrete domain of ints

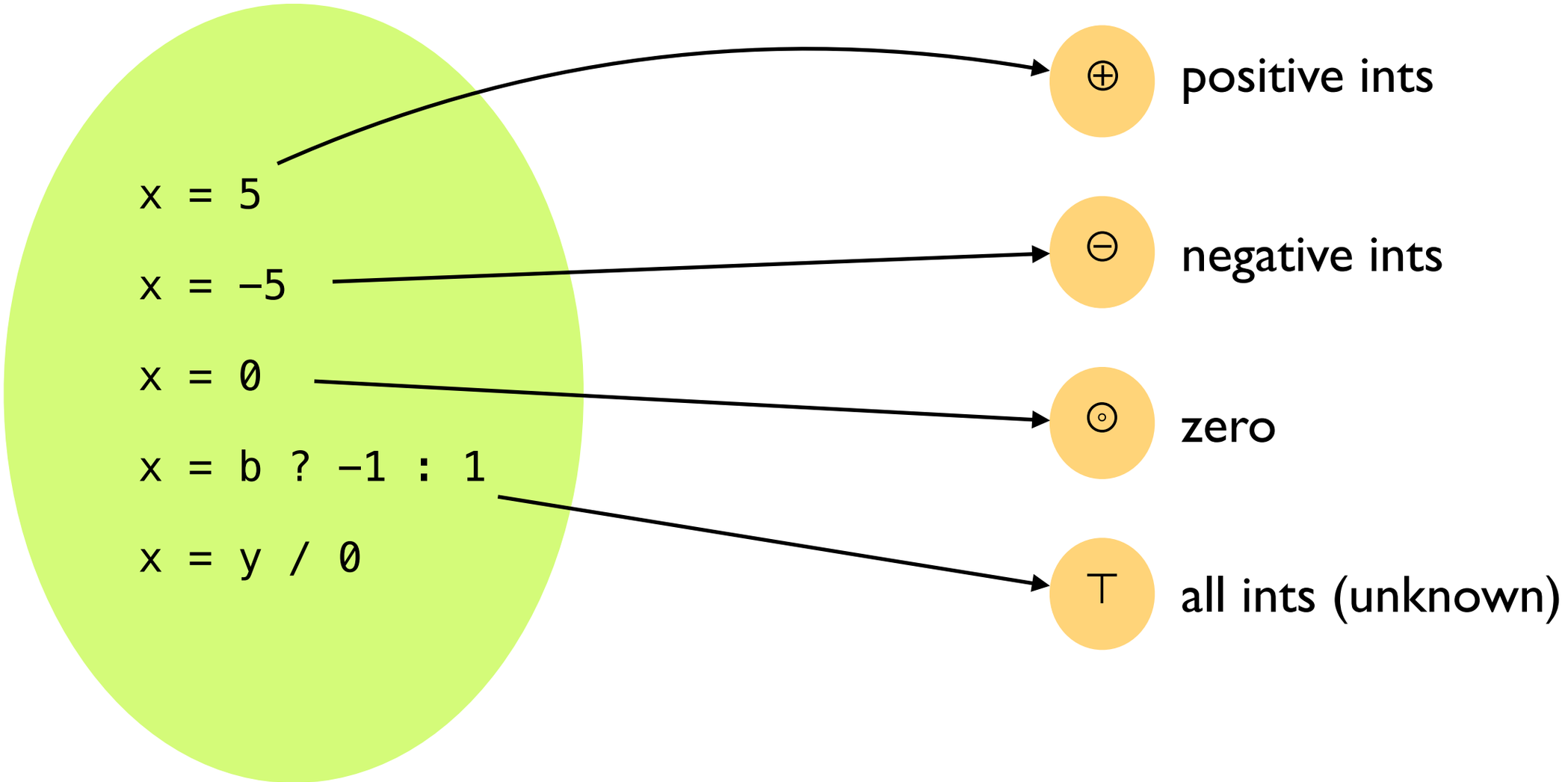
abstract domain of signs



A toy static analysis: abstraction

concrete domain of ints

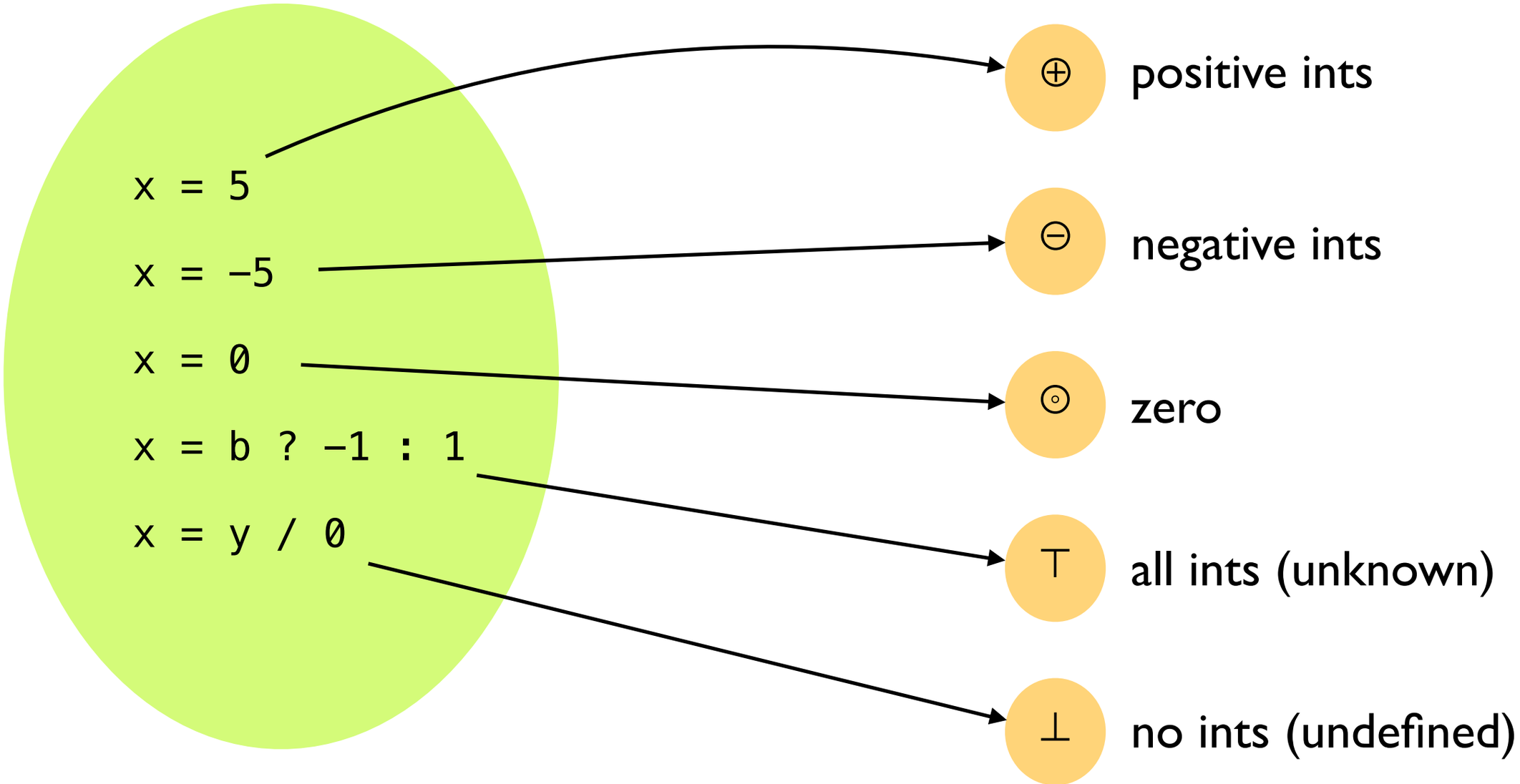
abstract domain of signs



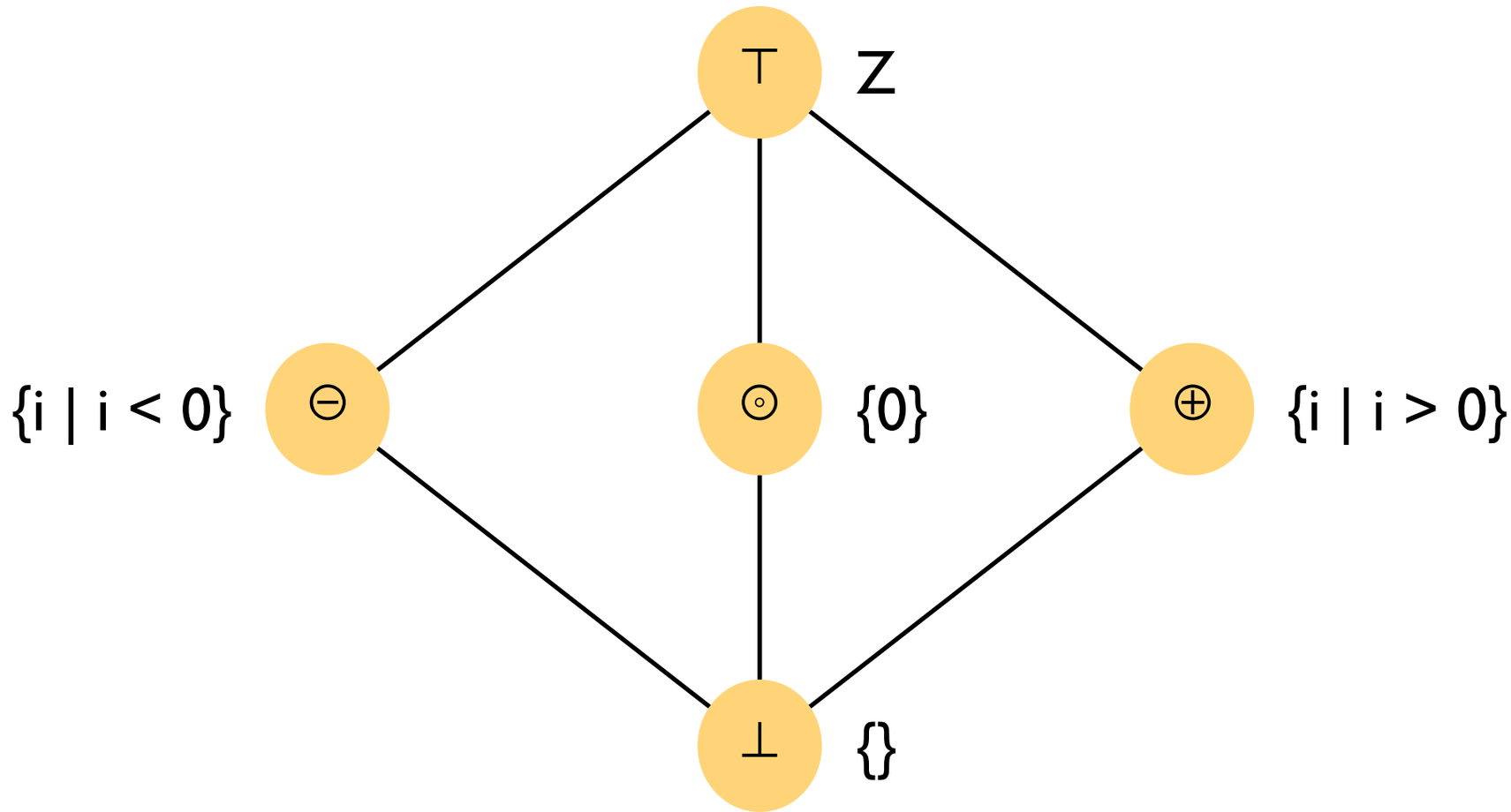
A toy static analysis: abstraction

concrete domain of ints

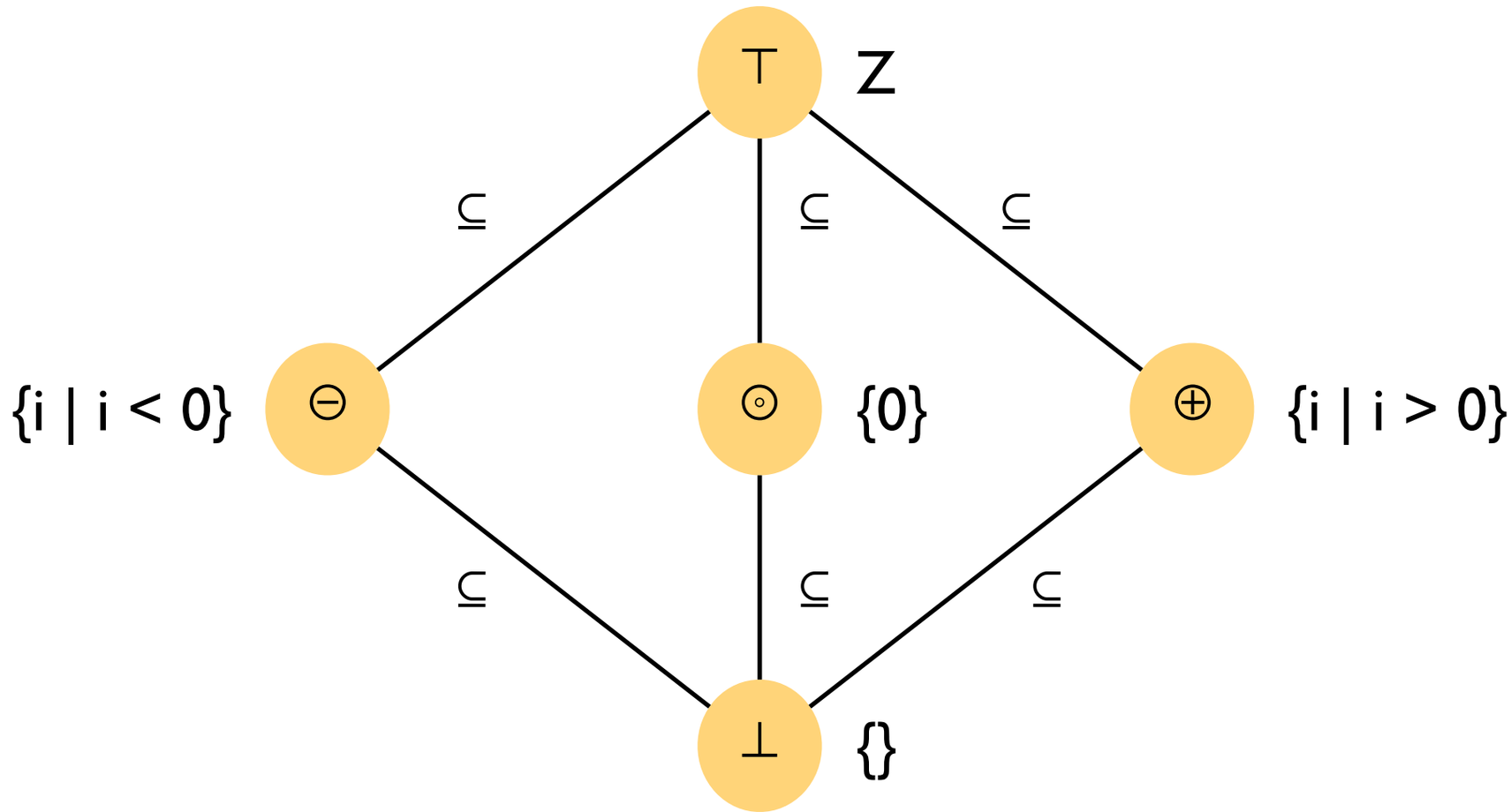
abstract domain of signs



A toy static analysis: abstraction



A toy static analysis: abstraction



A toy static analysis: transfer functions

- **Transfer functions** specify how to evaluate program expressions on abstract values.
  - $\oplus + \oplus =$
  - $\ominus + \ominus =$
  - $\odot + \odot =$
  - $\oplus + \ominus =$
  - $\top / \odot =$.
  - …

A toy static analysis: transfer functions

- **Transfer functions** specify how to evaluate program expressions on abstract values.
  - $\oplus + \oplus = \oplus$
  - $\ominus + \ominus =$
  - $\odot + \odot =$
  - $\oplus + \ominus =$
  - $\top / \odot =$.
  - …

A toy static analysis: transfer functions

- **Transfer functions** specify how to evaluate program expressions on abstract values.
  - $\oplus + \oplus = \oplus$
  - $\ominus + \ominus = \ominus$
  - $\odot + \odot =$
  - $\oplus + \ominus =$
  - $\top / \odot =$.
  - …

A toy static analysis: transfer functions

- **Transfer functions** specify how to evaluate program expressions on abstract values.
  - ⊕ + ⊕ = ⊕
  - ⊖ + ⊖ = ⊖
  - ⊙ + ⊙ = ⊙
  - ⊕ + ⊖ =
  - ⊤ / ⊙ = .
  - …

A toy static analysis: transfer functions

- **Transfer functions** specify how to evaluate program expressions on abstract values.
  - $\oplus + \oplus = \oplus$
  - $\ominus + \ominus = \ominus$
  - $\odot + \odot = \odot$
  - $\oplus + \ominus = \top$
  - $\top / \odot =$.
  - …

A toy static analysis: transfer functions

- **Transfer functions** specify how to evaluate program expressions on abstract values.
  - $\oplus + \oplus = \oplus$
  - $\ominus + \ominus = \ominus$
  - $\odot + \odot = \odot$
  - $\oplus + \ominus = \top$
  - $\top / \odot = \bot$
  - …

A toy static analysis: an example

```
a = 5;
b = -3;
c = a * b;
d = 0;
e = c * d;
f = 10 / e;
```

A toy static analysis: an example

```
a = ⊕;
b = -3;
c = a * b;
d = 0;
e = c * d;
f = 10 / e;
```

A toy static analysis: an example

```
a = ⊕;
b = ⊖;
c = a * b;
d = 0;
e = c * d;
f = 10 / e;
```

A toy static analysis: an example

```
a = ⊕;
b = ⊖;
c = ⊖;
d = 0;
e = c * d;
f = 10 / e;
```

A toy static analysis: an example

```
a = ⊕;
b = ⊖;
c = ⊖;
d = ⊙;
e = c * d;
f = 10 / e;
```

A toy static analysis: an example

```
a = ⊕;
b = ⊖;
c = ⊖;
d = ⊙;
e = ⊙;
f = 10 / e;
```

A toy static analysis: an example

```
a = ⊕;
b = ⊖;
c = ⊖;
d = ⊙;
e = ⊙;
f = ⊥;
```

A toy static analysis: an example

```
a = ⊕;
b = ⊖;
c = ⊖;
d = ⊙;
e = ⊙;
f = ⊥;
```

Detected division by zero!
Just look for variables that the analysis maps to ⊥.

A toy static analysis: another example

```
a = 5;
b = -3;
c = a + b;
d = 0;
e = c - d;
f = 10 / e;
```

A toy static analysis: another example

```
a = ⊕;
b = -3;
c = a + b;
d = 0;
e = c - d;
f = 10 / e;
```

A toy static analysis: another example

```
a = ⊕;
b = ⊖;
c = a + b;
d = 0;
e = c - d;
f = 10 / e;
```

A toy static analysis: another example

```
a = ⊕;
b = ⊖;
c = ⊤;
d = 0;
e = c - d;
f = 10 / e;
```

A toy static analysis: another example

```
a = ⊕;
b = ⊖;
c = ⊤;
d = ⊙;
e = c - d;
f = 10 / e;
```

A toy static analysis: another example

```
a = ⊕;
b = ⊖;
c = ⊤;
d = ⊙;
e = ⊤;
f = 10 / e;
```

A toy static analysis: another example

```
a = ⊕;
b = ⊖;
c = ⊤;
d = ⊙;
e = ⊤;
f = ⊤;
```

A toy static analysis: another example

```
a = ⊕;
b = ⊖;
c = ⊤;
d = ⊙;
e = ⊤;
f = ⊤;
```

False positive! This program can never throw an error, but the analysis reports that f may contain any value (including undefined).

state-of-the-art static analysis tools

Some state-of-the-art static analysis tools

- Astree
- Coverity
- Java PathFinder
- …

Astree (sound)

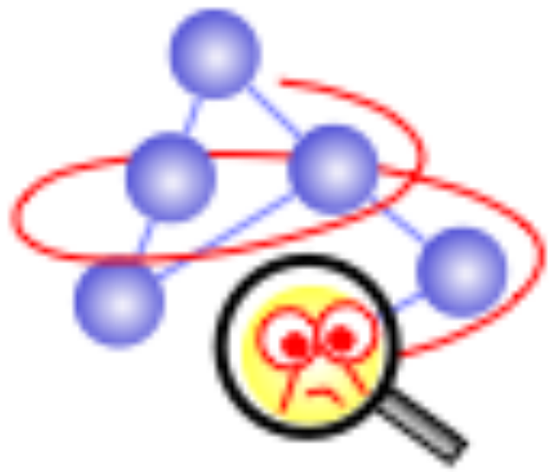
- Proves the absence of runtime errors and undefined behavior in C programs.
- Used to prove absence of runtime errors in
    - Airbus flight control software
    - Docking software for the International Space Station
- Many man-years of effort (since 2001) to develop.
- See www.astree.ens.fr/

Coverity (neither sound nor complete)

- Looks for bugs in C, C++, Java, and C#.
- Used by
    - >1100 companies.
    - NASA JPL (in addition to many other tools).
- Offered as a free, cloud-based service for open-source projects.
- See www.coverity.com

Java PathFinder (sound but can be imprecise)

- Finds bugs in mission-critical Java code.
- Developed by NASA.
- Focuses on concurrency errors (race conditions), uncaught exceptions.
- Free and open source!
- See babelfish.arc.nasa.gov/trac/jpf



Summary

- Static analysis tools check if a program P satisfies a property φ by
  - (sound) overapproximation of P
  - (complete) underapproximation of P
- Many uses from compilers to bug finding to verification.
- Many high-quality tools available.