Static Program Analysis

Yue Li and Tian Tan

2020 Spring

Static Program Analysis

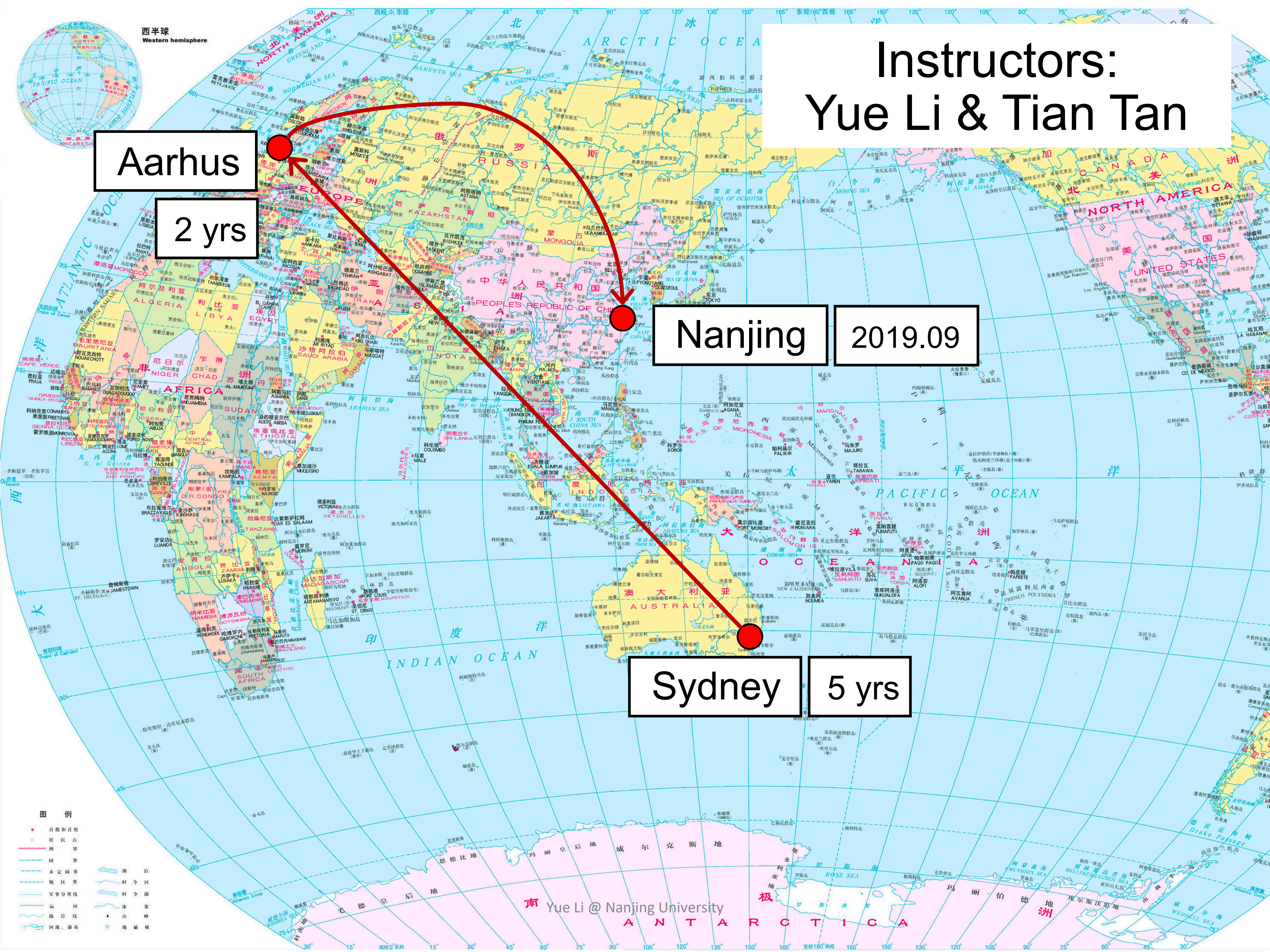
Introduction

Nanjing University

Yue Li

2020

Instructors: Yue Li & Tian Tan

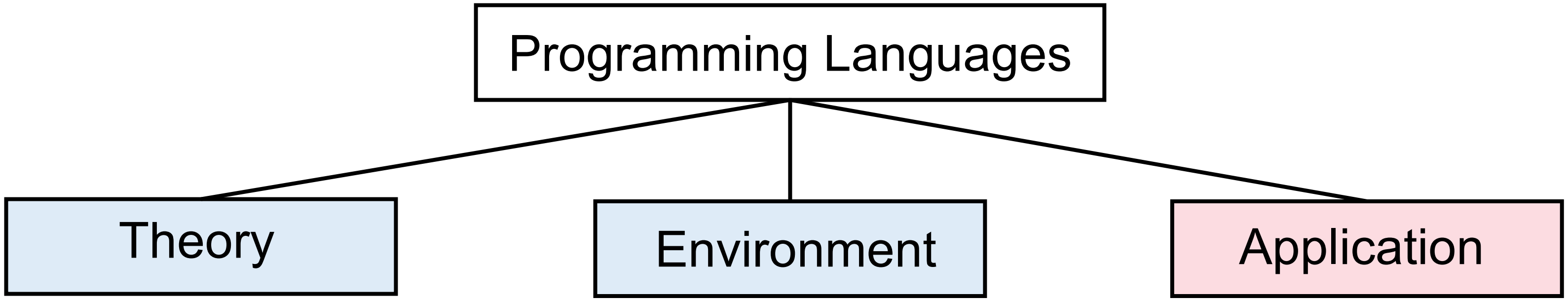


Contents

Static Program Analysis (Static Analysis)

Programming Languages

Static Program Analysis (Static Analysis)



- Language design
- Type system
- Semantics and logics
- … …

- Compilers
- Runtime system
- … …

- Program analysis
- Program verification
- Program synthesis
- … …

Static Program Analysis (Static Analysis)

Programming Languages

Theory
- Language design
- Type system
- Semantics and logics
- … …

Environment
- Compilers
- Runtime system
- … …

Application
- Program analysis
- Program verification
- Program synthesis
- … …

Background: In the last decade, the language cores had few changes, but the programs became significantly larger and more complicated.

Challenge: How to ensure the reliability, security and other promises of large-scale and complex programs?

Why We Need Static Analysis?

Why We Need Static Analysis?

- Program Reliability

  Null pointer dereference, memory leak, etc.

  Examples

Why We Need Static Analysis?

- Program Reliability

  Null pointer dereference, memory leak, etc.

  Examples

- Program Security

  Private information leak, injection attack, etc.

  Examples

Why We Need Static Analysis?

- Program Reliability

  Null pointer dereference, memory leak, etc.

  Examples

- Program Security

  Private information leak, injection attack, etc.

  Examples

- Compiler Optimization

  Dead code elimination, code motion, etc.

  Examples

Why We Need Static Analysis?

- Program Reliability

  Null pointer dereference, memory leak, etc.

  Examples

- Program Security

  Private information leak, injection attack, etc.

  Examples

- Compiler Optimization

  Dead code elimination, code motion, etc.

  Examples

- Program Understanding

  IDE call hierarchy, type indication, etc.

  Examples

Market of Static Analysis

Academia

Programming Languages

Software Engineering

Systems

Security

… …

Any directions that rely on programs

Industries

Microsoft

ORACLE

HUAWEI

facebook

IBM Research

GRAMMATECH

Uber

Semmle™

SAMSUNG

Google

Sourcebrella

Market of Static Analysis

Academia

Programming Languages

Software Engineering

Systems

Sec

… …

Any directions that
rely on programs

Industries

Microsoft

ORACLE

HUAWEI

book

Research

GRAMMATECH

Uber

Semmle™

SAMSUNG

Google

Sourcebrella

Static analysis people are urgently needed !

Static Analysis

Static analysis analyzes a program *P* to reason about its behaviors and determines whether it satisfies some properties before running *P*.

- Does *P* contain any private information leaks?
- Does *P* dereference any null pointers?
- Are all the cast operations in *P* safe?
- Can *v1* and *v2* in *P* point to the same memory location?
- Will certain *assert* statements in *P* fail?
- Is this piece of code in *P* dead (so that it could be eliminated)?
- …

Static Analysis

Static analysis analyzes a program *P* to reason about its behaviors and determines whether it satisfies some properties before running *P*.

- Does *P* contain any private information leaks?
- Does *P* dereference any null pointers?
- Are all the cast operations in *P* safe?
- Can *v1* and *v2* in *P* point to the same memory location?
- Will certain *assert* statements in *P* fail?
- Is this piece of code in *P* dead (so that it could be eliminated)?
- …

Unfortunately, by **Rice's Theorem**, there is no such approach to determine whether P satisfies such non-trivial properties, i.e., giving ***exact answer***: Yes or No

Rice's Theorem

"Any non-trivial property of the behavior of programs in a r.e. language is undecidable."

r.e. (recursively enumerable) = recognizable by a Turing-machine

Rice's Theorem

"Any non-trivial property of the behavior of programs in a r.e. language is undecidable."

r.e. (recursively enumerable) = recognizable by a Turing-machine

A property is trivial if either it is not satisfied by any r.e. language, or if it is satisfied by all r.e. languages; otherwise it is non-trivial.

non-trivial properties
~= interesting properties
~= the properties related with run-time behaviors of programs

Rice's Theorem

"Any non-trivial property of the behavior of programs in a r.e. language is undecidable."

r.e. (recursively enumerable) = recognizable by a Turing-machine

A property is trivial if either it is not satisfied by any r.e. language, or if it is satisfied by all r.e. languages; otherwise it is non-trivial.

non-trivial properties
~= interesting properties
~= the properties related with run-time behaviors of programs

- Does *P* contain any private information leaks?
- Does *P* dereference any null pointers?
- Are all the cast operation
- Can *v1* ar the same memory location?
- Will certain *assert* statements in *P* fail?
- Is this piece of code in *P* dead (so that it could be eliminated)?

Non-trivial Properties

Can determine whether P satisfies such non-trivial properties, i.e., giving ***exact answer***: Yes or No

Perfect static analysis

Can determine whether P satisfies such non-trivial properties, i.e., giving ***exact answer***: Yes or No

Perfect static analysis

Rice

Can determine whether P satisfies such non-trivial properties, i.e., giving ***exact answer***: Yes or No

Perfect static analysis

AND
- Sound
- Complete

Rice

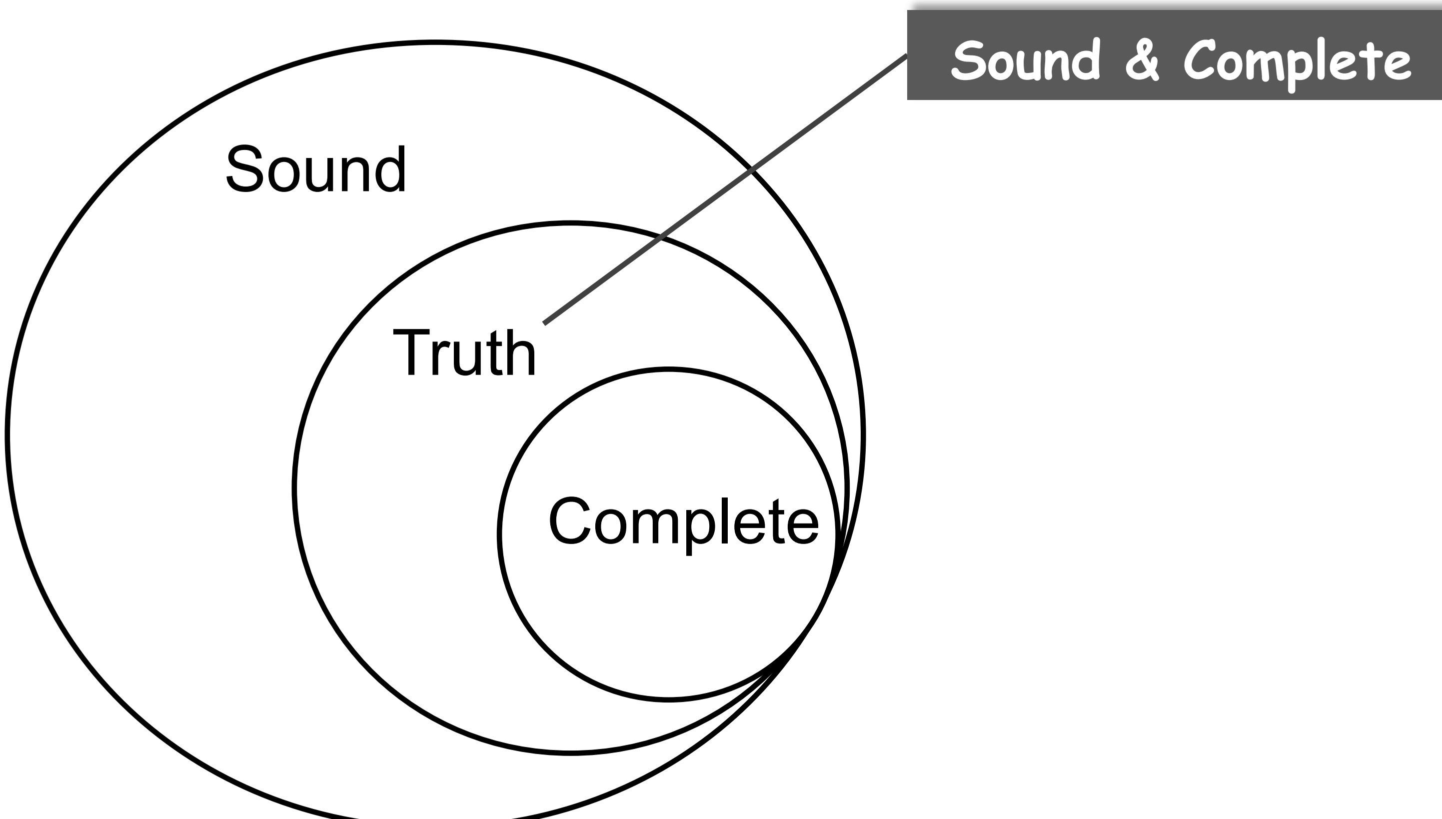
Can determine whether P satisfies such non-trivial properties, i.e., giving ***exact answer***: Yes or No

Perfect static analysis

AND

- Sound
- Complete

Rice



Can determine whether P satisfies such non-trivial properties, i.e., giving ***exact answer***: Yes or No

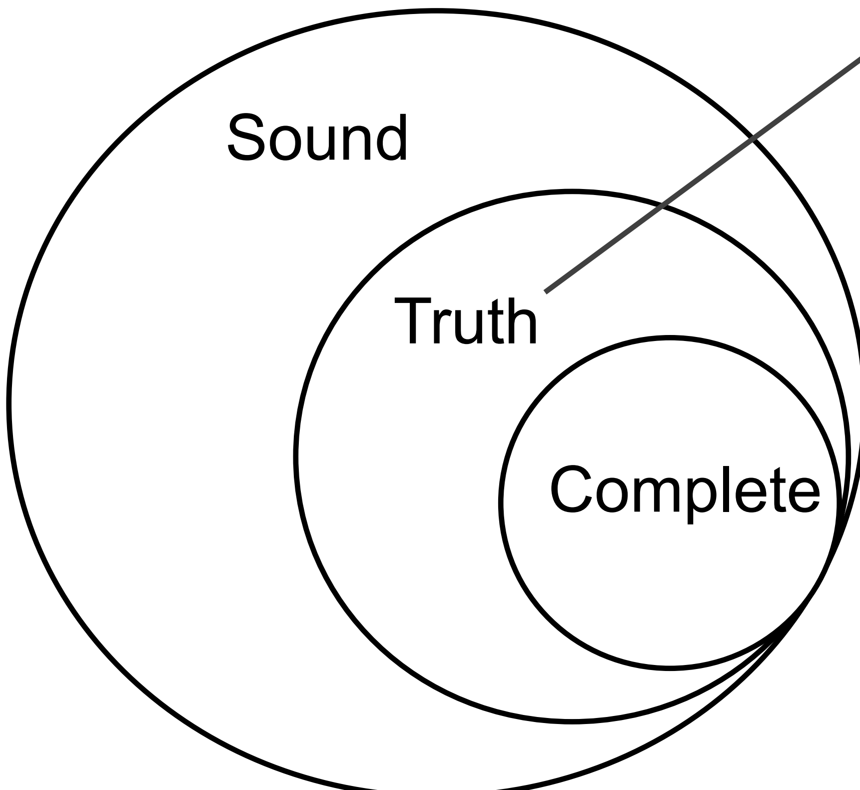
Perfect static analysis

AND
- Sound
- Complete

Rice

Sound & Complete

All possible true program behaviors

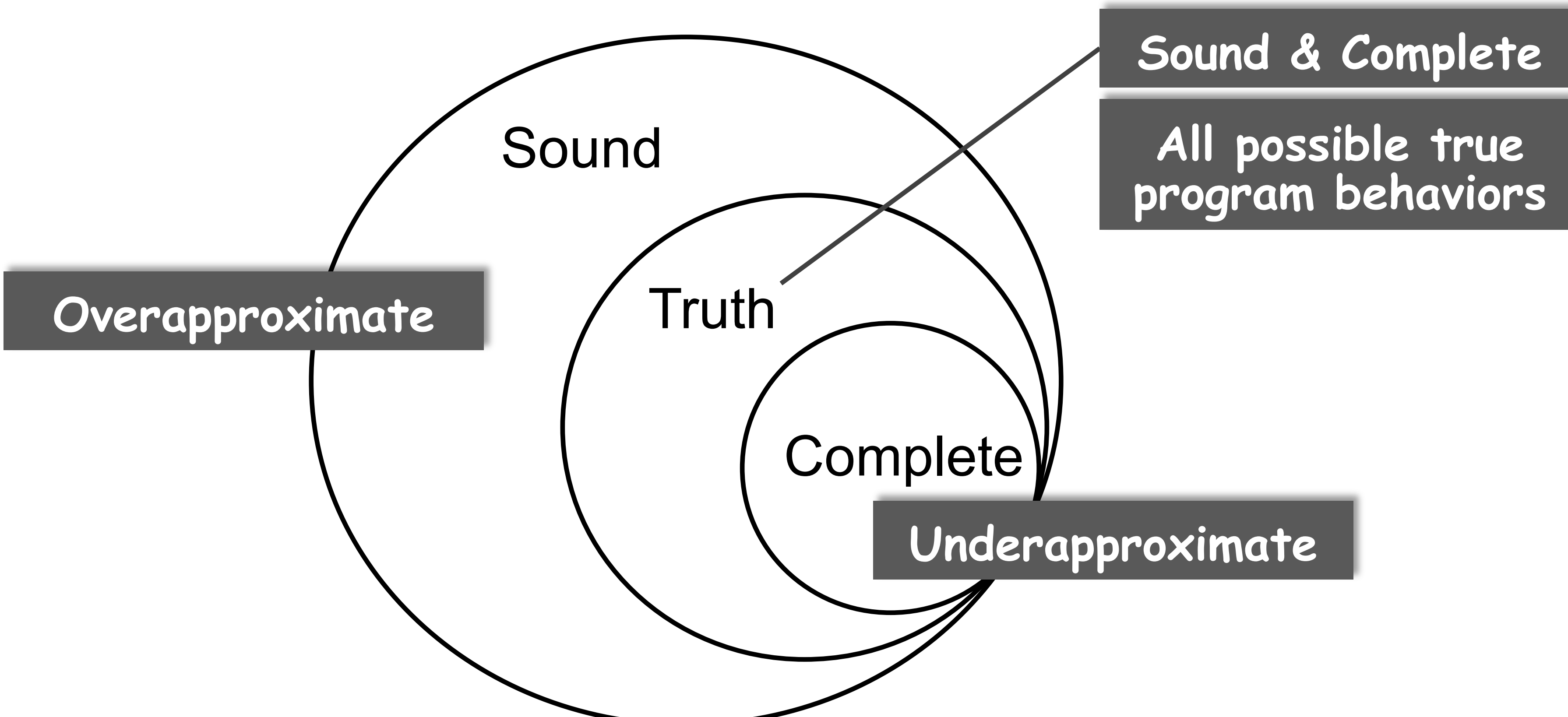


Can determine whether P satisfies such non-trivial properties, i.e., giving ***exact answer***: Yes or No

Perfect static analysis

AND
- Sound
- Complete

Rice



Can determine whether P satisfies such non-trivial properties, i.e., giving ***exact answer***: Yes or No

Perfect static analysis

AND

- Sound
- Complete

Rice

Sound & Complete

All possible true ... am behaviors

Sound

NO perfect static analysis!

The end of story ???

Complete

Underapproximate

Perfect static analysis

Rice

AND

- Sound
- Complete

Useful static analysis

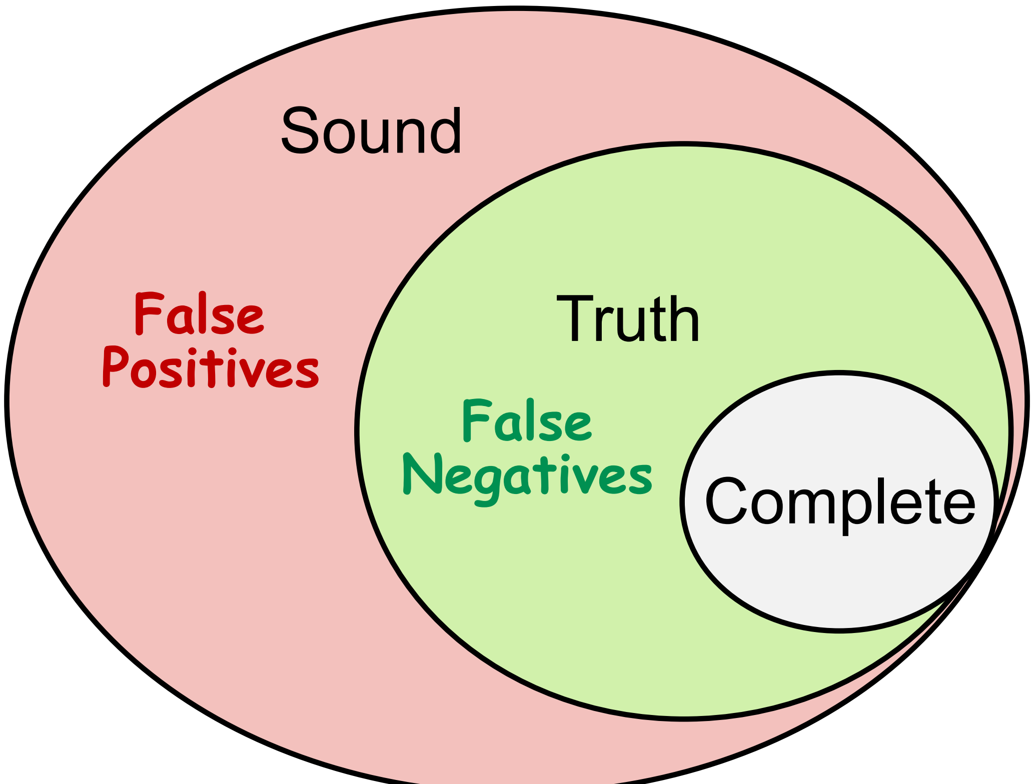
OR

- Compromise soundness (false negatives)
- Compromise completeness (false positives)

Useful static analysis

OR

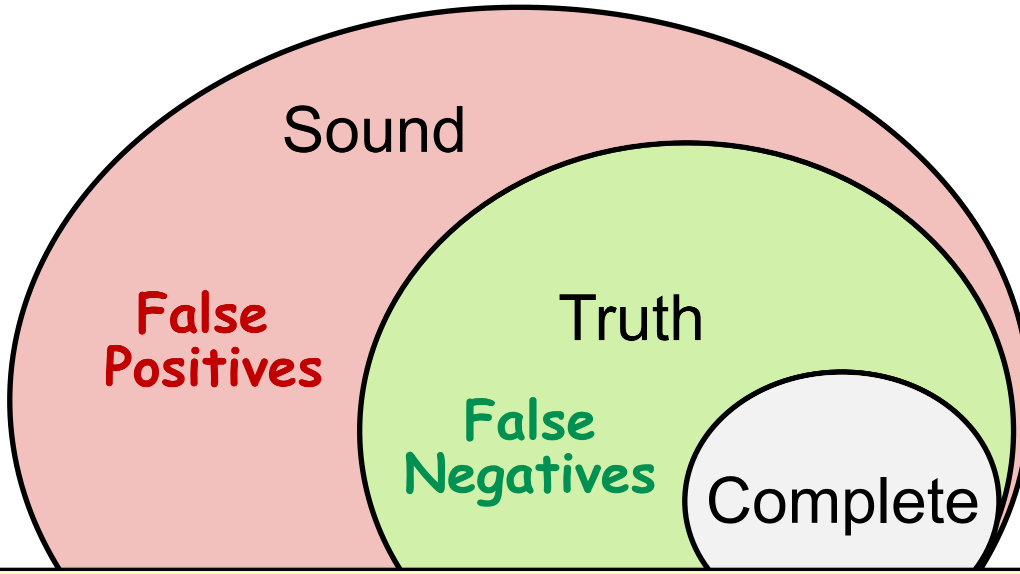
- Compromise soundness (false negatives)
- Compromise completeness (false positives)



Useful static analysis

OR

- Compromise soundness (false negatives)
- Compromise completeness (false positives)



Mostly compromising completeness:
Sound but not fully-precise static analysis

Necessity of Soundness

- Soundness is critical to a collection of important (static-analysis) applications such as *compiler optimization* and *program verification.*

Necessity of Soundness

- Soundness is critical to a collection of important (static-analysis) applications such as *compiler optimization* and *program verification.*

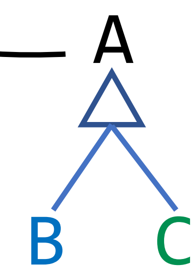
Unsound

"Safe Cast": wrong conclusion

```
B b = new B();
a.fld = b;
```

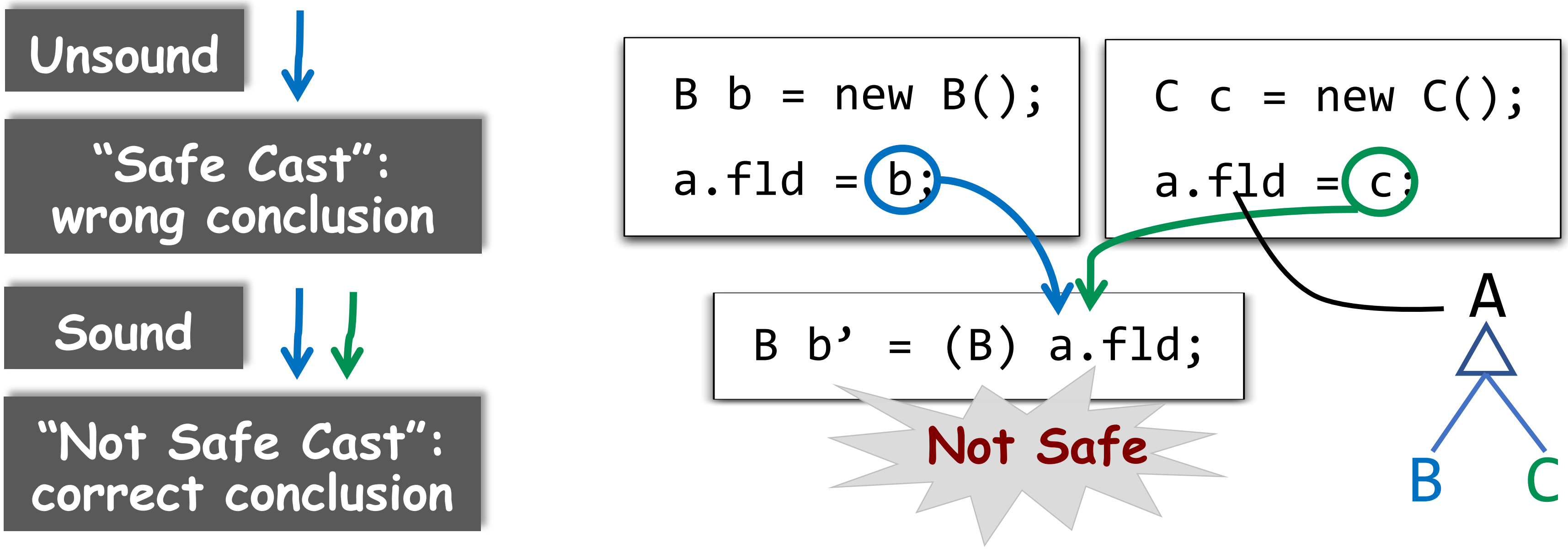
```
C c = new C();
a.fld = c;
```

```
B b' = (B) a.fld;
```



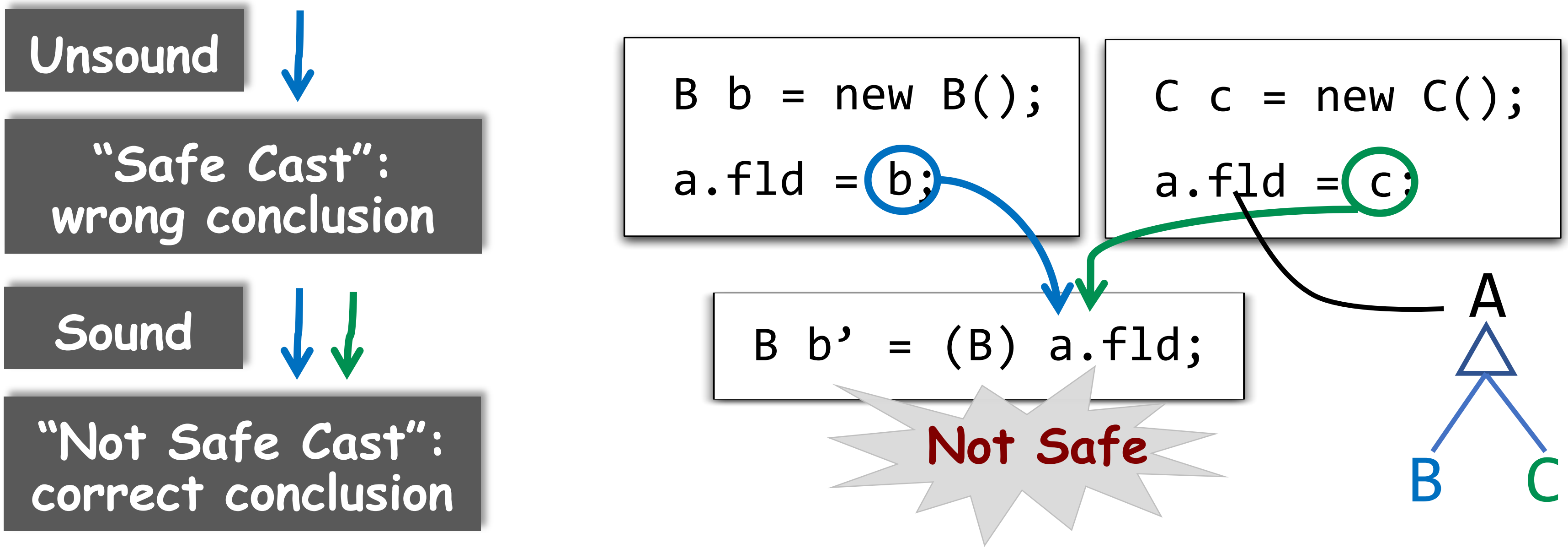
Necessity of Soundness

- Soundness is critical to a collection of important (static-analysis) applications such as *compiler optimization* and *program verification.*



Necessity of Soundness

- Soundness is critical to a collection of important (static-analysis) applications such as *compiler optimization* and *program verification.*



- Soundness is also preferable to other (static-analysis) applications for which soundness is not demanded, e.g., *bug detection*, as better soundness implies more bugs could be found.

Static Analysis — Bird's Eye View

```
if(input)
  x = 1;
else
  x = 0;
→ x = ?
```

Static Analysis — Bird's Eye View

```
if(input)
    x = 1;
else
    x = 0;
→ x = ?
```

Two analysis results:

1. when `input` is *true*, x = 1
   when `input` is *false*, x = 0

2. x = 1 or x = 0

Static Analysis — Bird's Eye View

```
    if(input)
       x = 1;
    else
       x = 0;
→ x = ?
```

Two analysis results:

1. when `input` is *true*, x = 1
   when `input` is *false*, x = 0

   Sound, precise, expensive

2. x = 1 or x = 0

   Sound, imprecise, cheap

Static Analysis — Bird's Eye View

```
if(input)
   x = 1;
else
   x = 0;
→ x = ?
```

Two analysis results:

1. when `input` is *true*, x = 1
   when `input` is *false*, x = 0

   Sound, precise, expensive

2. x = 1 or x = 0

   Sound, imprecise, cheap

Static Analysis: ensure (or get close to) soundness, while making good trade-offs between analysis precision and analysis speed.

For most static analyses
(may analysis)

Two Words to Conclude Static Analysis

Abstraction + **Over-approximation**

Static Analysis — An Example

Determine the sign (+, -, or 0) of all the variables of a given program.

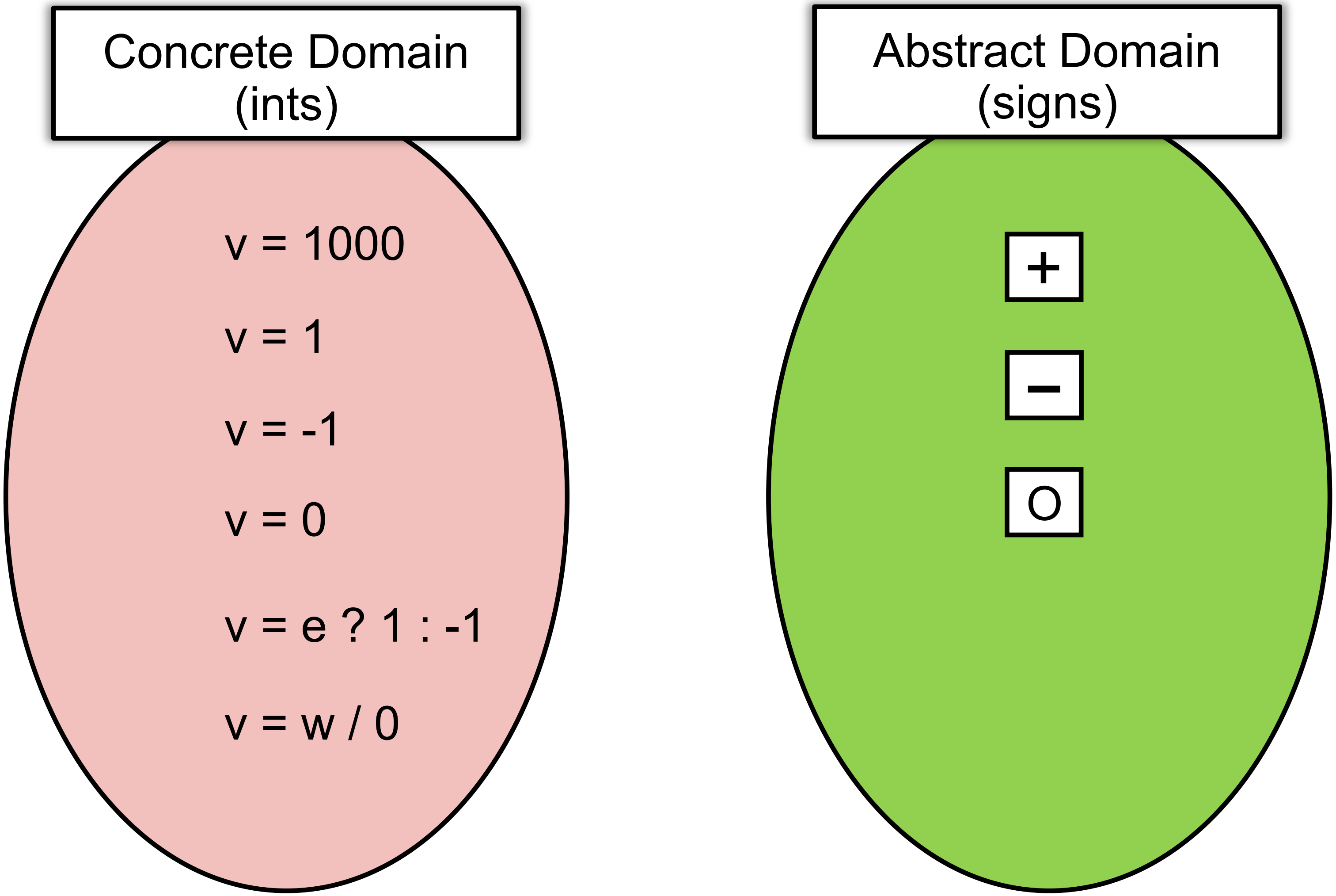
- Abstraction
- Over-approximation
  - Transfer functions
  - Control flows

To check divided by zero error

To check negative array indices

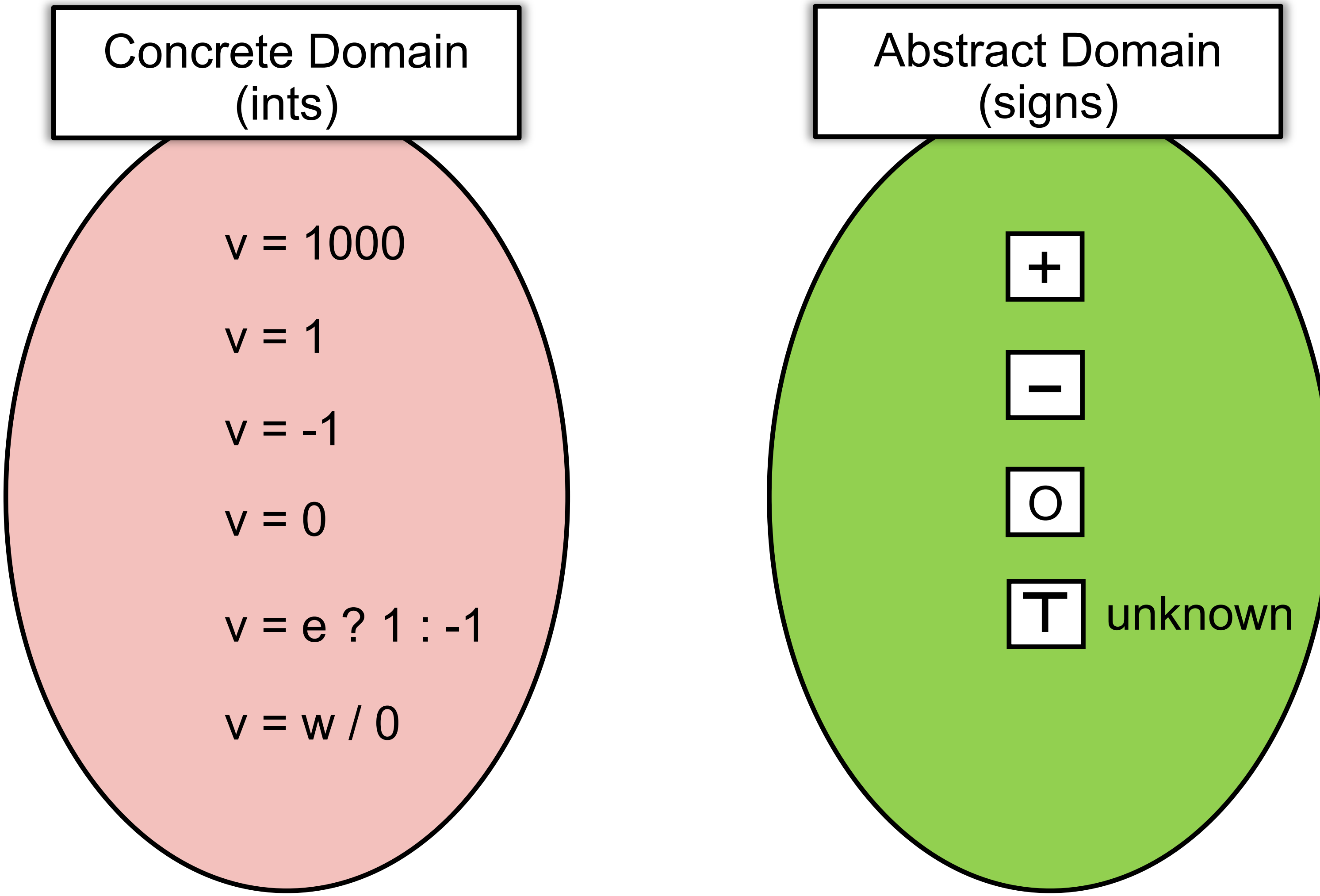
Abstraction

Determine the sign (+, -, or 0) of all the variables of a given program.



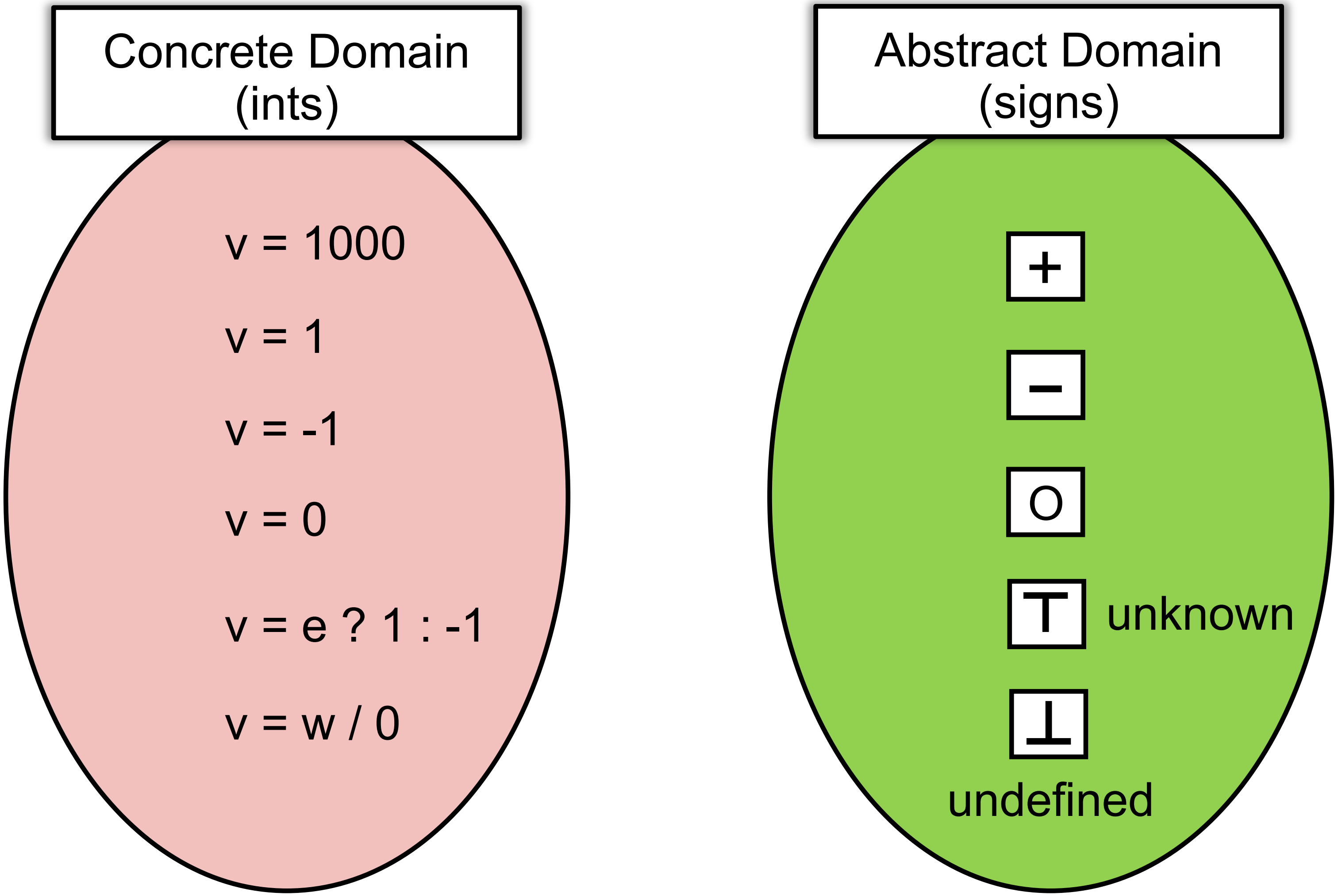
Abstraction

Determine the sign (+, -, or 0) of all the variables of a given program.



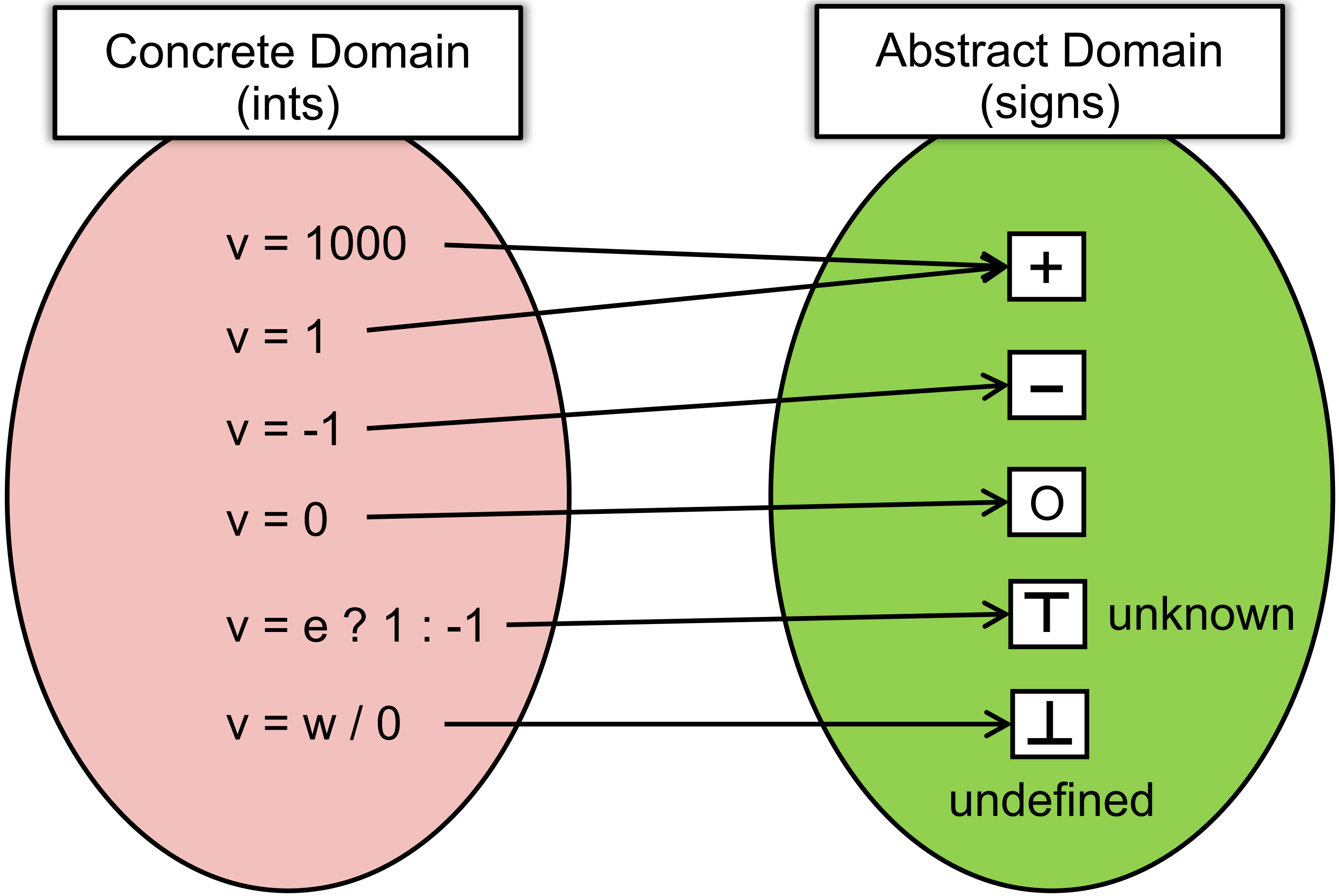
Abstraction

Determine the sign (+, -, or 0) of all the variables of a given program.



Abstraction

Determine the sign (+, -, or 0) of all the variables of a given program.



Over-approximation: Transfer Functions

- In static analysis, transfer functions define how to evaluate different program statements on abstract values.

- Transfer functions are defined according to “analysis problem” and the “semantics” of different program statements.

Over-approximation: Transfer Functions

- In static analysis, transfer functions define how to evaluate different program statements on abstract values.
- Transfer functions are defined according to “analysis problem” and the “semantics” of different program statements.

$\boxed{+} + \boxed{+} =$	$\boxed{+} \,/\, \boxed{+} =$
$\boxed{-} + \boxed{-} =$	$\boxed{-} \,/\, \boxed{-} =$
$\boxed{0} + \boxed{0} =$	$\boxed{\top} \,/\, \boxed{0} =$
$\boxed{+} + \boxed{-} =$	$\boxed{+} \,/\, \boxed{-} =$

Over-approximation: Transfer Functions

- In static analysis, transfer functions define how to evaluate different program statements on abstract values.
- Transfer functions are defined according to “analysis problem” and the “semantics” of different program statements.

$\boxed{+} + \boxed{+} = \boxed{+}$

$\boxed{-} + \boxed{-} = \boxed{-}$

$\boxed{0} + \boxed{0} = \boxed{0}$

$\boxed{+} + \boxed{-} =$

$\boxed{+} \,/\, \boxed{+} = \boxed{+}$

$\boxed{-} \,/\, \boxed{-} = \boxed{+}$

$\boxed{\top} \,/\, \boxed{0} =$

$\boxed{+} \,/\, \boxed{-} = \boxed{-}$

Over-approximation: Transfer Functions

- In static analysis, transfer functions define how to evaluate different program statements on abstract values.
- Transfer functions are defined according to "analysis problem" and the "semantics" of different program statements.

$$\boxed{+} + \boxed{+} = \boxed{+} \qquad \boxed{+} / \boxed{+} = \boxed{+}$$

$$\boxed{-} + \boxed{-} = \boxed{-} \qquad \boxed{-} / \boxed{-} = \boxed{+}$$

$$\boxed{0} + \boxed{0} = \boxed{0} \qquad \boxed{\top} / \boxed{0} =$$

$$\boxed{+} + \boxed{-} = \boxed{\top} \qquad \boxed{+} / \boxed{-} = \boxed{-}$$

Over-approximation: Transfer Functions

- In static analysis, transfer functions define how to evaluate different program statements on abstract values.
- Transfer functions are defined according to “analysis problem” and the “semantics” of different program statements.

$$\boxed{+} + \boxed{+} = \boxed{+} \qquad \boxed{+} \,/\, \boxed{+} = \boxed{+}$$

$$\boxed{-} + \boxed{-} = \boxed{-} \qquad \boxed{-} \,/\, \boxed{-} = \boxed{+}$$

$$\boxed{0} + \boxed{0} = \boxed{0} \qquad \boxed{\top} \,/\, \boxed{0} = \boxed{\bot}$$

$$\boxed{+} + \boxed{-} = \boxed{\top} \qquad \boxed{+} \,/\, \boxed{-} = \boxed{-}$$

$$\boxed{+} + \boxed{+} = \boxed{+}$$

$$\boxed{-} + \boxed{-} = \boxed{-}$$

$$\boxed{\mathrm{O}} + \boxed{\mathrm{O}} = \boxed{\mathrm{O}}$$

$$\boxed{+} + \boxed{-} = \boxed{\top}$$

$$\boxed{+} \;/\; \boxed{+} = \boxed{+}$$

$$\boxed{-} \;/\; \boxed{-} = \boxed{+}$$

$$\boxed{\top} \;/\; \boxed{\mathrm{O}} = \boxed{\bot}$$

$$\boxed{+} \;/\; \boxed{-} = \boxed{-}$$

$$\boxed{\mathrm{O}} \;/\; \boxed{-} = \boxed{\mathrm{O}}$$

… …

$\boxed{+} + \boxed{+} = \boxed{+}$

$\boxed{-} + \boxed{-} = \boxed{-}$

$\boxed{0} + \boxed{0} = \boxed{0}$

$\boxed{+} + \boxed{-} = \boxed{\top}$

$\boxed{+} \;/\; \boxed{+} = \boxed{+}$

$\boxed{-} \;/\; \boxed{-} = \boxed{+}$

$\boxed{\top} \;/\; \boxed{0} = \boxed{\bot}$

$\boxed{+} \;/\; \boxed{-} = \boxed{-}$

$\boxed{0} \;/\; \boxed{-} = \boxed{0}$

... ...

```
x = 10;
y = -1;
z = 0;
a = x + y;
b = z / y;
c = a / b;
p = arr[y];
q = arr[a];
```

```
x =
y =
z =
a =
b =
c =
p =
q =
```

$\boxed{+} + \boxed{+} = \boxed{+}$

$\boxed{-} + \boxed{-} = \boxed{-}$

$\boxed{0} + \boxed{0} = \boxed{0}$

$\boxed{+} + \boxed{-} = \boxed{\top}$

$\boxed{+} / \boxed{+} = \boxed{+}$

$\boxed{-} / \boxed{-} = \boxed{+}$

$\boxed{\top} / \boxed{0} = \boxed{\bot}$

$\boxed{+} / \boxed{-} = \boxed{-}$

$\boxed{0} / \boxed{-} = \boxed{0}$

… …

```
x = 10;
y = -1;
z = 0;
a = x + y;
b = z / y;
c = a / b;
p = arr[y];
q = arr[a];
```

$x = \boxed{+}$

$y = \boxed{-}$

$z = \boxed{0}$

a =

b =

c =

p =

q =

$\boxed{+} + \boxed{+} = \boxed{+}$

$\boxed{-} + \boxed{-} = \boxed{-}$

$\boxed{O} + \boxed{O} = \boxed{O}$

$\boxed{+} + \boxed{-} = \boxed{\top}$

$\boxed{+} / \boxed{+} = \boxed{+}$

$\boxed{-} / \boxed{-} = \boxed{+}$

$\boxed{\top} / \boxed{O} = \boxed{\bot}$

$\boxed{+} / \boxed{-} = \boxed{-}$

$\boxed{O} / \boxed{-} = \boxed{O}$

... ...

```
x = 10;
y = -1;
z = 0;
a = x + y;
b = z / y;
c = a / b;
p = arr[y];
q = arr[a];
```

$x = \boxed{+}$

$y = \boxed{-}$

$z = \boxed{O}$

$a = \boxed{\top}$

$b =$

$c =$

$p =$

$q =$

$\boxed{+} + \boxed{+} = \boxed{+}$

$\boxed{-} + \boxed{-} = \boxed{-}$

$\boxed{0} + \boxed{0} = \boxed{0}$

$\boxed{+} + \boxed{-} = \boxed{\top}$

$\boxed{+} / \boxed{+} = \boxed{+}$

$\boxed{-} / \boxed{-} = \boxed{+}$

$\boxed{\top} / \boxed{0} = \boxed{\bot}$

$\boxed{+} / \boxed{-} = \boxed{-}$

$\boxed{0} / \boxed{-} = \boxed{0}$

… …

```
x = 10;
y = -1;
z = 0;
a = x + y;
b = z / y;
c = a / b;
p = arr[y];
q = arr[a];
```

$x = \boxed{+}$

$y = \boxed{-}$

$z = \boxed{0}$

$a = \boxed{\top}$

$b = \boxed{0}$

c =

p =

q =

$\boxed{+} + \boxed{+} = \boxed{+}$

$\boxed{-} + \boxed{-} = \boxed{-}$

$\boxed{0} + \boxed{0} = \boxed{0}$

$\boxed{+} + \boxed{-} = \boxed{\top}$

$\boxed{+} / \boxed{+} = \boxed{+}$

$\boxed{-} / \boxed{-} = \boxed{+}$

$\boxed{\top} / \boxed{0} = \boxed{\bot}$

$\boxed{+} / \boxed{-} = \boxed{-}$

$\boxed{0} / \boxed{-} = \boxed{0}$

… …

```
x = 10;
y = -1;
z = 0;
a = x + y;
b = z / y;
c = a / b;
p = arr[y];
q = arr[a];
```

$x = \boxed{+}$

$y = \boxed{-}$

$z = \boxed{0}$

$a = \boxed{\top}$

$b = \boxed{0}$

$c = \boxed{\bot}$

p =

q =

$\boxed{+} + \boxed{+} = \boxed{+}$

$\boxed{-} + \boxed{-} = \boxed{-}$

$\boxed{0} + \boxed{0} = \boxed{0}$

$\boxed{+} + \boxed{-} = \boxed{\top}$

$\boxed{+} \,/\, \boxed{+} = \boxed{+}$

$\boxed{-} \,/\, \boxed{-} = \boxed{+}$

$\boxed{\top} \,/\, \boxed{0} = \boxed{\bot}$

$\boxed{+} \,/\, \boxed{-} = \boxed{-}$

$\boxed{0} \,/\, \boxed{-} = \boxed{0}$

... ...

```
x = 10;
y = -1;
z = 0;
a = x + y;
b = z / y;
c = a / b;
p = arr[y];
q = arr[a];
```

$x = \boxed{+}$

$y = \boxed{-}$

$z = \boxed{0}$

$a = \boxed{\top}$

$b = \boxed{0}$

$c = \boxed{\bot}$

$p = \boxed{\bot}$

$q =$

$$\boxed{+} + \boxed{+} = \boxed{+}$$

$$\boxed{-} + \boxed{-} = \boxed{-}$$

$$\boxed{0} + \boxed{0} = \boxed{0}$$

$$\boxed{+} + \boxed{-} = \boxed{\top}$$

$$\boxed{+} \,/\, \boxed{+} = \boxed{+}$$

$$\boxed{-} \,/\, \boxed{-} = \boxed{+}$$

$$\boxed{\top} \,/\, \boxed{0} = \boxed{\bot}$$

$$\boxed{+} \,/\, \boxed{-} = \boxed{-}$$

$$\boxed{0} \,/\, \boxed{-} = \boxed{0}$$

… …

```
x = 10;
y = -1;
z = 0;
a = x + y;
b = z / y;
c = a / b;
p = arr[y];
q = arr[a];
```

$x = \boxed{+}$

$y = \boxed{-}$

$z = \boxed{0}$

$a = \boxed{\top}$

$b = \boxed{0}$

$c = \boxed{\bot}$

$p = \boxed{\bot}$

$q = \boxed{\bot}$

$\boxed{+} + \boxed{+} = \boxed{+}$

$\boxed{-} + \boxed{-} = \boxed{-}$

$\boxed{0} + \boxed{0} = \boxed{0}$

$\boxed{+} + \boxed{-} = \boxed{\top}$

$\boxed{+} / \boxed{+} = \boxed{+}$

$\boxed{-} / \boxed{-} = \boxed{+}$

$\boxed{\top} / \boxed{0} = \boxed{\bot}$

$\boxed{+} / \boxed{-} = \boxed{-}$

$\boxed{0} / \boxed{-} = \boxed{0}$

… …

```
x = 10;
y = -1;
z = 0;
a = x + y;
b = z / y;
c = a / b;
p = arr[y];
q = arr[a];
```

1

$x = \boxed{+}$

$y = \boxed{-}$

$z = \boxed{0}$

$a = \boxed{\top}$

$b = \boxed{0}$

$c = \boxed{\bot}$ Divided by zero

$p = \boxed{\bot}$

$q = \boxed{\bot}$

$\boxed{+} + \boxed{+} = \boxed{+}$

$\boxed{-} + \boxed{-} = \boxed{-}$

$\boxed{0} + \boxed{0} = \boxed{0}$

$\boxed{+} + \boxed{-} = \boxed{\top}$

$\boxed{+} / \boxed{+} = \boxed{+}$

$\boxed{-} / \boxed{-} = \boxed{+}$

$\boxed{\top} / \boxed{0} = \boxed{\bot}$

$\boxed{+} / \boxed{-} = \boxed{-}$

$\boxed{0} / \boxed{-} = \boxed{0}$

… …

```
  x = 10;
  y = -1;
  z = 0;
  a = x + y;
  b = z / y;
1 c = a / b;
2 p = arr[y];
3 q = arr[a];
```

$x = \boxed{+}$

$y = \boxed{-}$

$z = \boxed{0}$

$a = \boxed{\top}$

$b = \boxed{0}$

$c = \boxed{\bot}$ Divided by zero

$p = \boxed{\bot}$

$q = \boxed{\bot}$ negative array index

$\boxed{+} + \boxed{+} = \boxed{+}$

$\boxed{-} + \boxed{-} = \boxed{-}$

$\boxed{0} + \boxed{0} = \boxed{0}$

$\boxed{+} + \boxed{-} = \boxed{\top}$

$\boxed{+} / \boxed{+} = \boxed{+}$

$\boxed{-} / \boxed{-} = \boxed{+}$

$\boxed{\top} / \boxed{0} = \boxed{\bot}$

$\boxed{+} / \boxed{-} = \boxed{-}$

$\boxed{0} / \boxed{-} = \boxed{0}$

… …

```
    x = 10;
    y = -1;
    z = 0;
    a = x + y;
    b = z / y;
(1) c = a / b;
(2) p = arr[y];
(3) q = arr[a];
```

$x = \boxed{+}$

$y = \boxed{-}$

$z = \boxed{0}$

$a = \boxed{\top}$

$b = \boxed{0}$

$c = \boxed{\bot}$ Divided by zero

$p = \boxed{\bot}$

$q = \boxed{\bot}$ negative array index

1 2 **Static analysis is useful**

$\boxed{+} + \boxed{+} = \boxed{+}$

$\boxed{-} + \boxed{-} = \boxed{-}$

$\boxed{0} + \boxed{0} = \boxed{0}$

$\boxed{+} + \boxed{-} = \boxed{\top}$

$\boxed{+} / \boxed{+} = \boxed{+}$

$\boxed{-} / \boxed{-} = \boxed{+}$

$\boxed{\top} / \boxed{0} = \boxed{\bot}$

$\boxed{+} / \boxed{-} = \boxed{-}$

$\boxed{0} / \boxed{-} = \boxed{0}$

… …

```
  x = 10;
  y = -1;
  z = 0;
  a = x + y;
  b = z / y;
1 c = a / b;
2 p = arr[y];
3 q = arr[a];
```

$x = \boxed{+}$

$y = \boxed{-}$

$z = \boxed{0}$

$a = \boxed{\top}$

$b = \boxed{0}$

$c = \boxed{\bot}$ Divided by zero

$p = \boxed{\bot}$

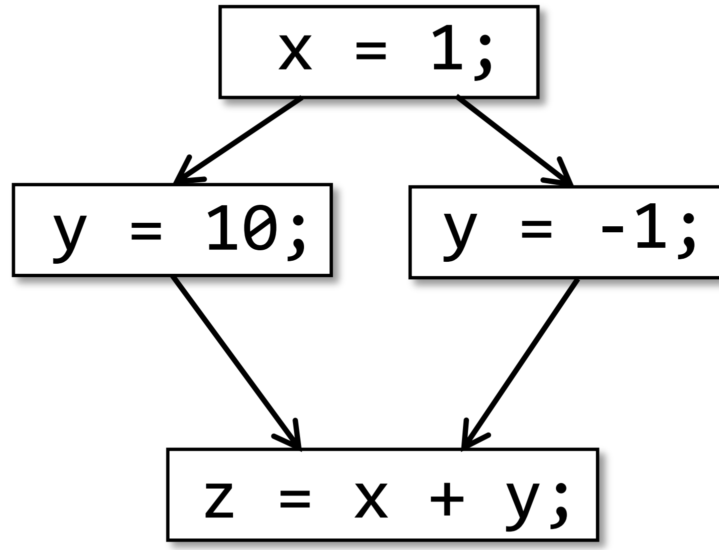
$q = \boxed{\bot}$ negative array index

1 2 **Static analysis is useful**

3 **But (over-approximated) static analysis produces false positives**

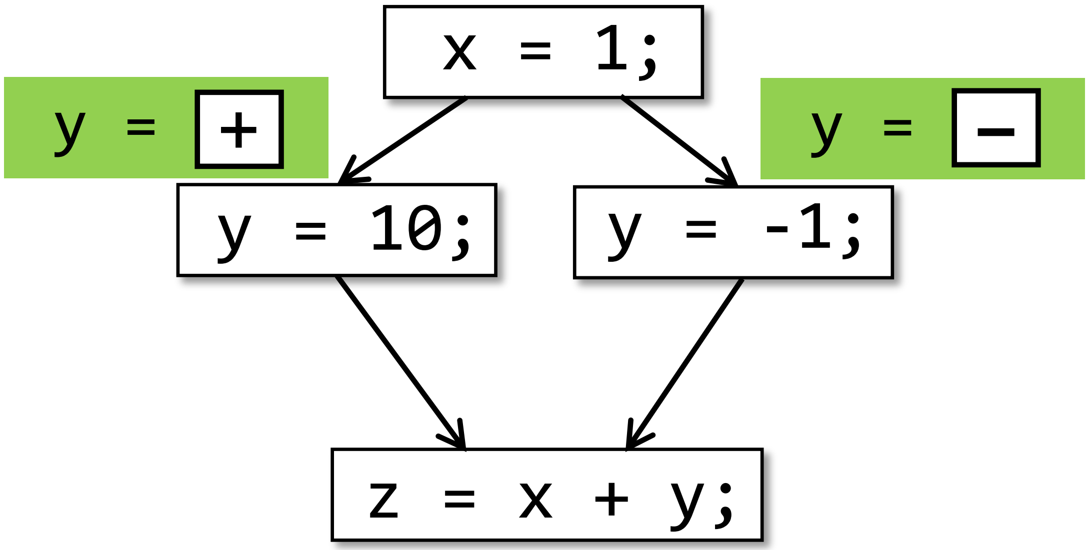
Over-approximation: Control Flows

```
x = 1;
if(input)
  y = 10;
else
  y = -1;
z = x + y;
```



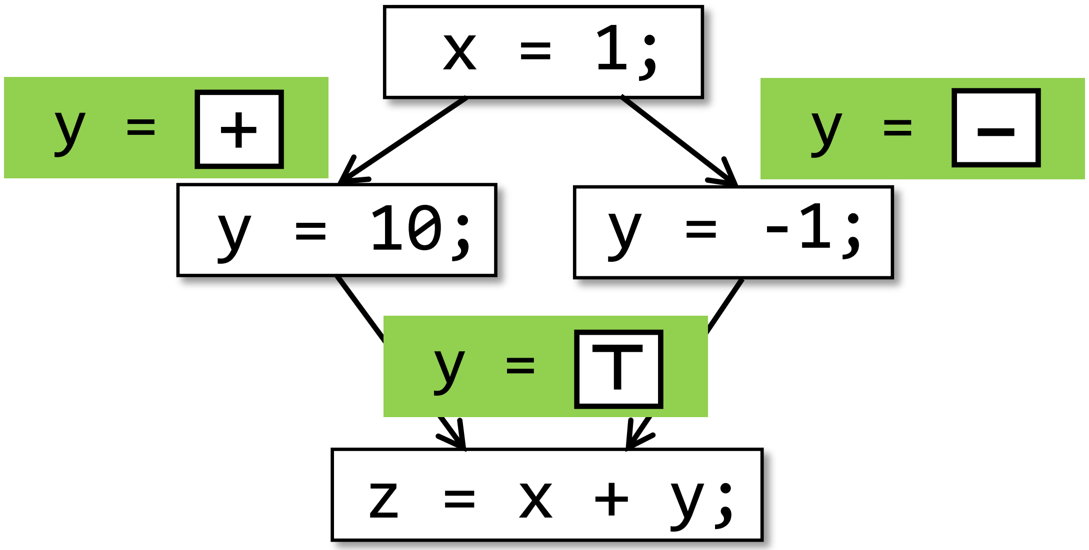
Over-approximation: Control Flows

```
x = 1;
if(input)
  y = 10;
else
  y = -1;
z = x + y;
```



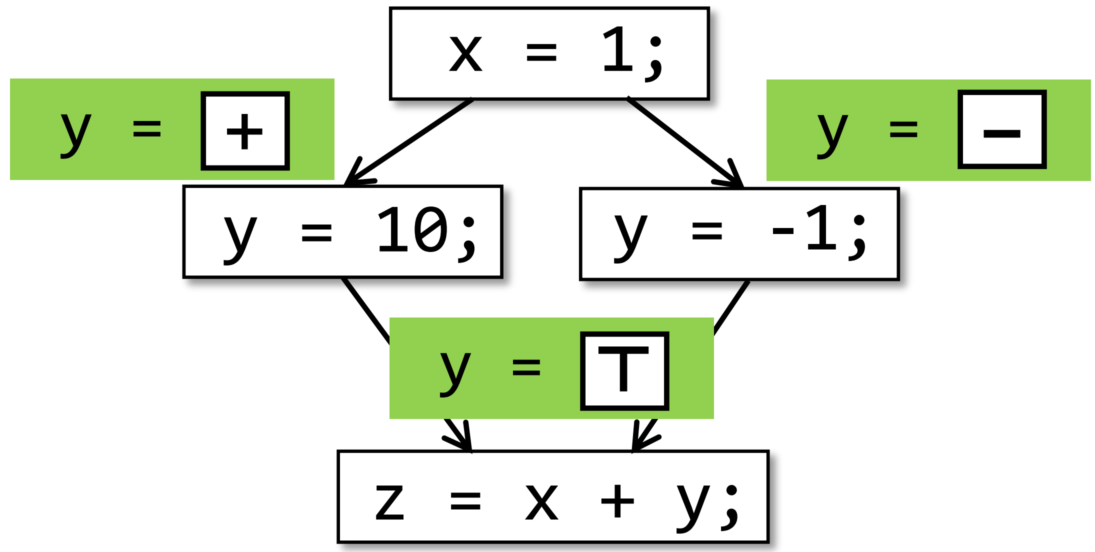
Over-approximation: Control Flows

```
x = 1;
if(input)
  y = 10;
else
  y = -1;
z = x + y;
```



Over-approximation: Control Flows

```
x = 1;
if(input)
  y = 10;
else
  y = -1;
z = x + y;
```



As it's impossible to enumerate all paths in practice, flow merging (as a way of over-approximation) is taken for granted in most static analyses.

Teaching Plan (Tentative)

1. Introduction
2. Intermediate Representation
3. Data Flow Analysis – Applications (I)
4. Data Flow Analysis – Applications (II)
5. Data Flow Analysis – Foundations (I)
6. Data Flow Analysis – Foundations (II)
7. Inter-procedural Analysis
8. Pointer Analysis
9. Pointer Analysis – Foundations (I)
10. Pointer Analysis – Foundations (II)
11. Pointer Analysis – Context Sensitivity
12. Static Analysis for Security
13. CFL-Reachability and IFDS
14. Soundness and Soundiness
15. Abstract Interpretation
16. Course Summary

Evaluation Criteria

- Coding Assignments 50%
- Final Exam 50%

Coding Assignments (Tentative)

- Assignment 1: Constant Propagation (CP, 10 points)
  - Statically compute and propagate constant values in program
  - Intra-procedural analysis

Coding Assignments (Tentative)

- Assignment 1: Constant Propagation (CP, 10 points)
  - Statically compute and propagate constant values in program
  - Intra-procedural analysis
- Assignment 2: Dead Code Elimination (DCE , 14 points)
  - Based on constant propagation, eliminate dead code in program
  - b =true; if (b) { … } else { /* dead code */ }

Coding Assignments (Tentative)

- Assignment 1: Constant Propagation (CP, 10 points)
  - Statically compute and propagate constant values in program
  - Intra-procedural analysis
- Assignment 2: Dead Code Elimination (DCE , 14 points)
  - Based on constant propagation, eliminate dead code in program
  - b =true; if (b) { … } else { /* dead code */ }
- Assignment 3: Class Hierarchy Analysis (CHA, 8 points)
  - Build a call graph via class hierarchy analysis
  - Enable inter-procedural constant propagation

Coding Assignments (Tentative)

- Assignment 1: Constant Propagation (CP, 10 points)
  - Statically compute and propagate constant values in program
  - Intra-procedural analysis
- Assignment 2: Dead Code Elimination (DCE , 14 points)
  - Based on constant propagation, eliminate dead code in program
  - b =true; if (b) { … } else { /* dead code */ }
- Assignment 3: Class Hierarchy Analysis (CHA, 8 points)
  - Build a call graph via class hierarchy analysis
  - Enable inter-procedural constant propagation
- Assignment 4: Pointer Analysis (PTA, 12 points)
  - Build a call graph via pointer analysis (more precise than CHA)
  - Enable more precise inter-procedural constant propagation

Coding Assignments (Tentative)

- Assignment 1: Constant Propagation (CP, 10 points)
  - Statically compute and propagate constant values in program
  - Intra-procedural analysis
- Assignment 2: Dead Code Elimination (DCE , 14 points)
  - Based on constant propagation, eliminate dead code in program
  - b =true; if (b) { … } else { /* dead code */ }
- Assignment 3: Class Hierarchy Analysis (CHA, 8 points)
  - Build a call graph via class hierarchy analysis
  - Enable inter-procedural constant propagation
- Assignment 4: Pointer Analysis (PTA, 12 points)
  - Build a call graph via pointer analysis (more precise than CHA)
  - Enable more precise inter-procedural constant propagation
- Assignment 5: Context-Sensitive Pointer Analysis (CSPTA, 6 points)
  - Build a call graph via C.S. pointer analysis (more precise than PTA)
  - Enable more precise inter-procedural constant propagation

The X You Need To Understand in This Lecture

- What are the differences between static analysis and (dynamic) testing?
- Understand soundness, completeness, false negatives, and false positives.
- Why soundness is ~~usually~~ required by static analysis?
- How to understand abstraction and over-approximation?

Our PASCAL Research Group @Nanjing University

PASCAL

Programming LAnguages and StatiC AnaLysis Group

Home | People | Publications | Code

The **PASCAL Research Group** is affiliated with Institute of Computer Software and Department of Computer Science and Technology at Nanjing University. We develop effective static program analysis techniques and tools for solving the problems in programming languages, software engineering, system and security.

News

October 15, 2019
Yue Li and Tian Tan start the **PASCAL Research Group** at Nanjing University!

PASCAL

Older posts...

People

Chenxi Zhang
Ph.D., 2017 — (co-supervised with Prof. Chang Xu)

Ganlin Li
Undergraduate, 2018 —

Hao Ling
Undergraduate, 2016 —

Shengyuan Yang
Undergraduate, 2017 —

Tian Tan
Assistant Research Professor

Weiyu Ye
Ph.D., 2017 — (co-supervised with Prof. Xiaoxing Ma)

Yue Li
Associate Professor

Yuying Yuan
Undergraduate, 2017 —


進撃の巨人
Season 2


536
李 樾
Yue Li
谭 添
Tian Tan

软件分析

南京大学

计算机科学与技术系

程序设计语言与

静态分析研究组

李樾 谭添