

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

Static Program Analysis

Introduction

Nanjing University

Yue Li

Fall 2022

Lecturers:

Yue Li & Tian Tan

Aarhus

2 yrs

Nanjing

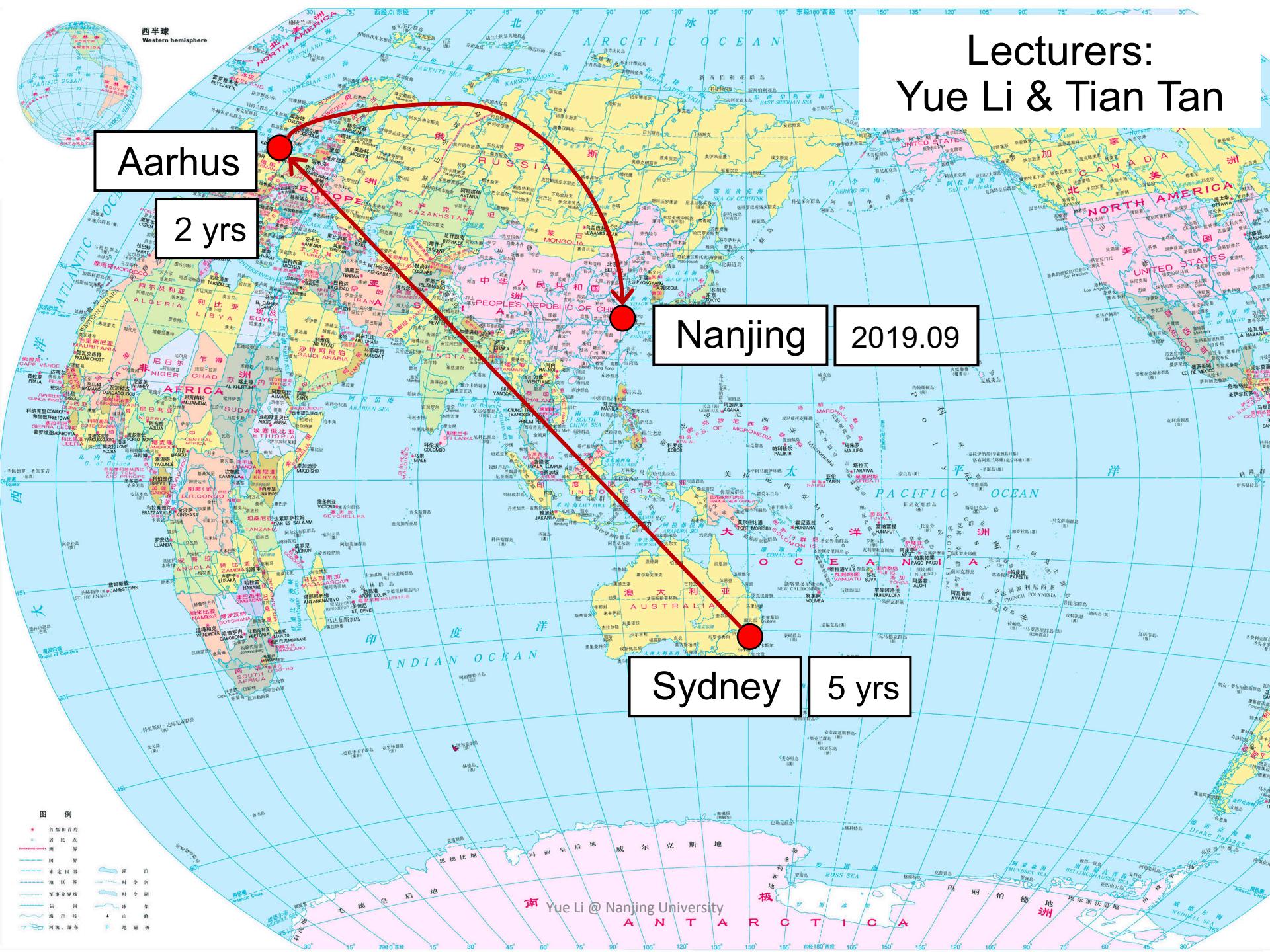
2019.09

Sydney

5 yrs



西半球
Western hemisphere



图例

- 首都和省会
- 国家和民族
- 国界
- - - 未定国界
- 地区界
- 军事界线
- 航运界线
- 河流、海流
- ▲ 山峰
- 湖 泊
- 时令河
- 冰架带
- 海岸线
- 河流、瀑布
- 地 碑

Lecturers:

Yue Li & Tian Tan

Aarhus

2 yrs

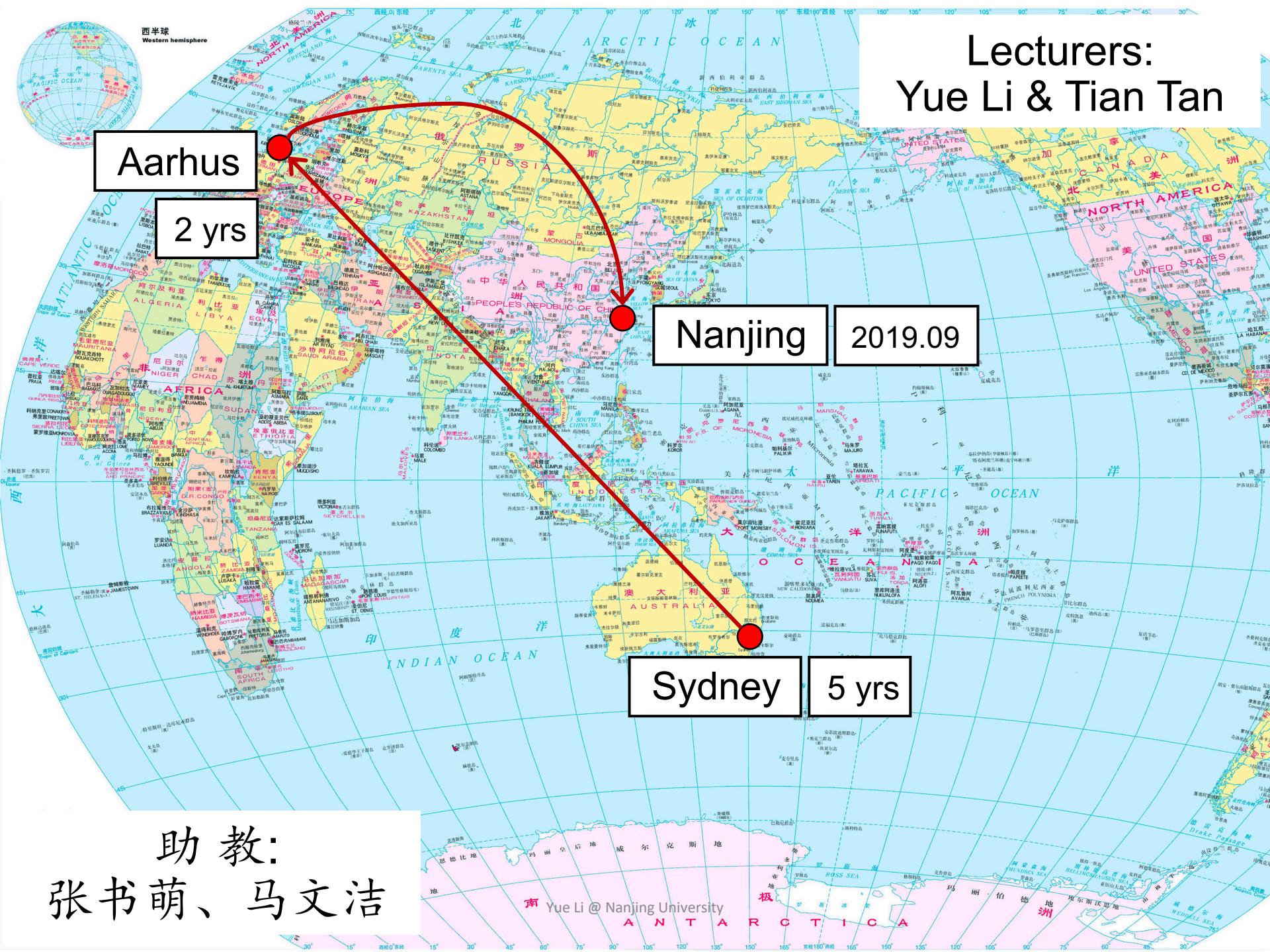
Nanjing

2019.09

Sydney

5 yrs

助教:
张书萌、马文洁



Contents

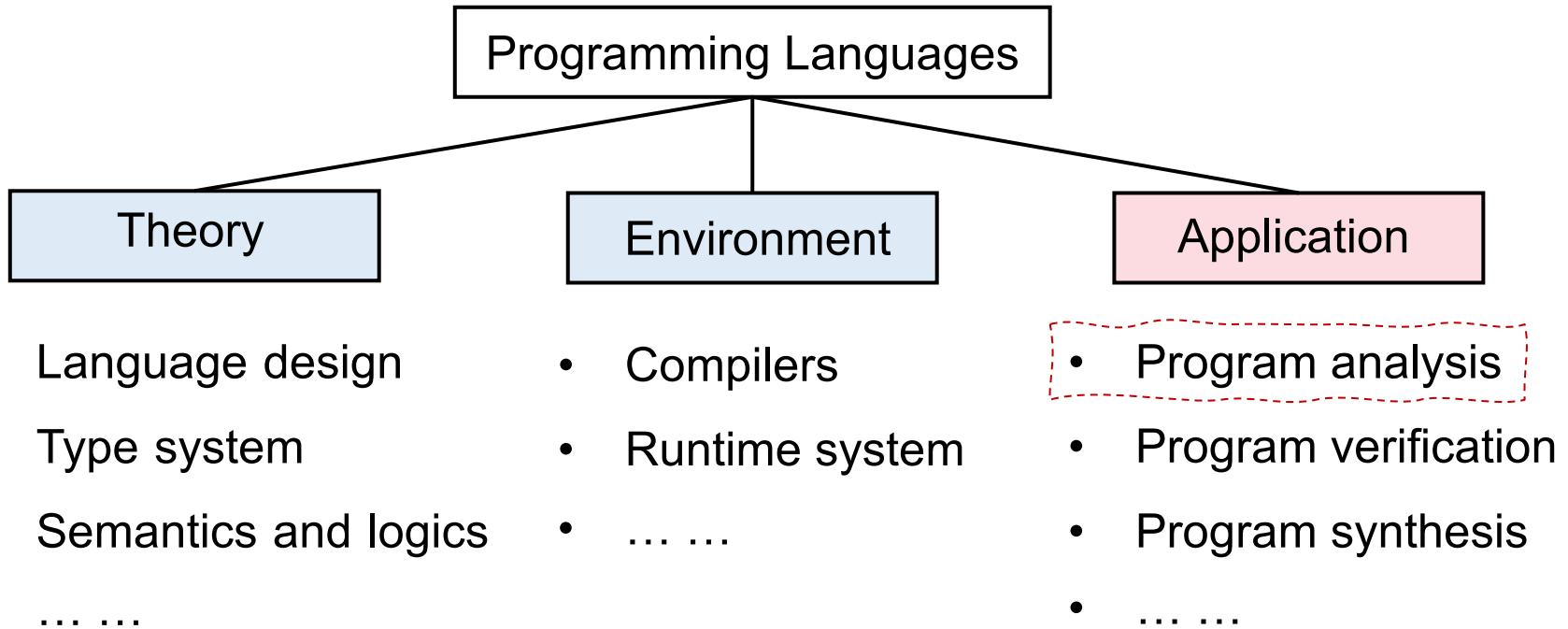
1. PL and Static Analysis
2. Why We Learn Static Analysis?
3. What is Static Analysis?
4. Static Analysis Features and Examples
5. Teaching Plan
6. Evaluation Criteria



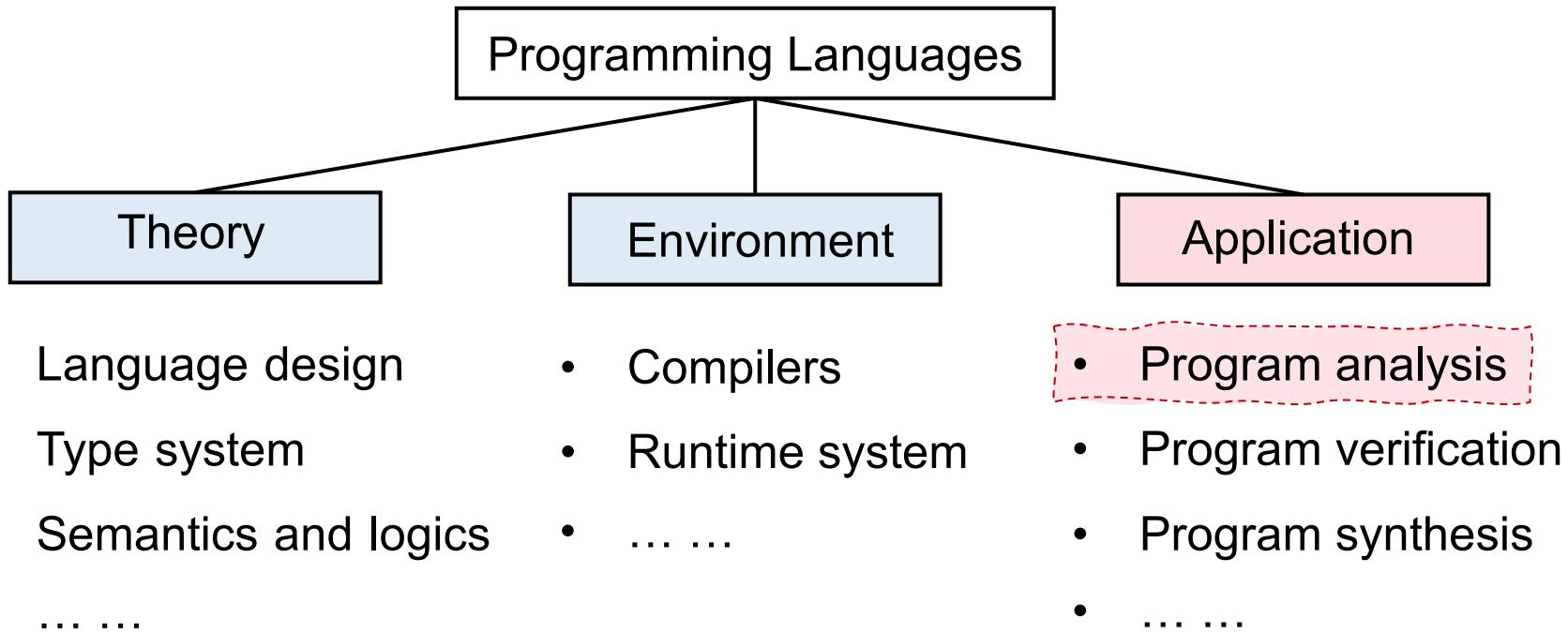
Static Program Analysis (Static Analysis)

Programming Languages

Static Program Analysis (Static Analysis)



Static Program Analysis (Static Analysis)



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.
- Program Security
 - Private information leak, injection attack, etc.
- Compiler Optimization
 - Dead code elimination, code motion, etc.

Examples

Examples

Examples

Why We Need Static Analysis?

- Program Reliability
 - Null pointer dereference, memory leak, etc.
- Program Security
 - Private information leak, injection attack, etc.
- Compiler Optimization
 - Dead code elimination, code motion, etc.
- Program Understanding
 - IDE call hierarchy, type indication, etc.

Market of Static Analysis

Academia

Programming Languages

Software Engineering

Systems

Security

....

Any directions that
rely on programs

Industries



Microsoft



facebook

IBM Research



Uber Semmle™



Market of Static Analysis

Academia

Programming Languages

Software Engineering

Systems

Security

....

Any directions that
rely on programs

Industries



Microsoft



HUAWEI



GRAMMATECH



Uber



Semmle™



SAMSUNG



Sourcebrella

Oracle, Google, Microsoft, IBM, Facebook, ...

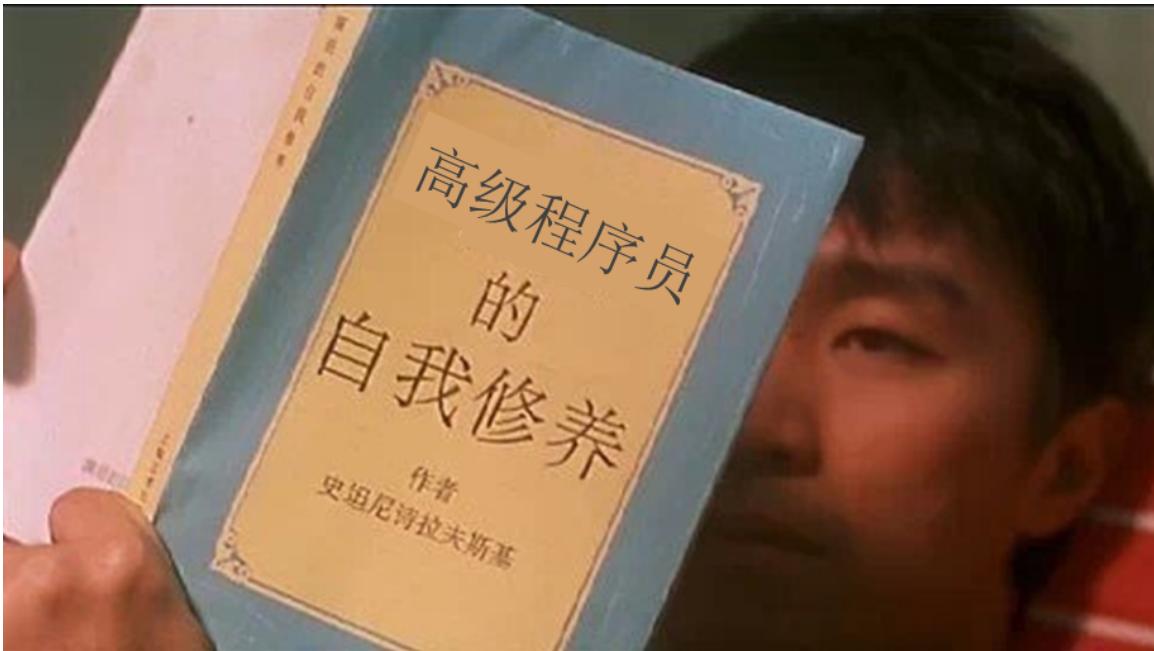
华为、阿里巴巴、腾讯、网易、字节跳动、京东、
航空航天研究院、国家电网研究院、中国电信等等



前方中文预警

深入学习静态程序分析——附加值

- 更深入地理解编程语言的语法、语义（不枯燥）
- 自然而然地写出更可靠、更安全、更高效的程序



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 operations legal?
- Can $v1$ and $v2$ both 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)?

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



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

Perfect static analysis

AND

- Sound
- Complete



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

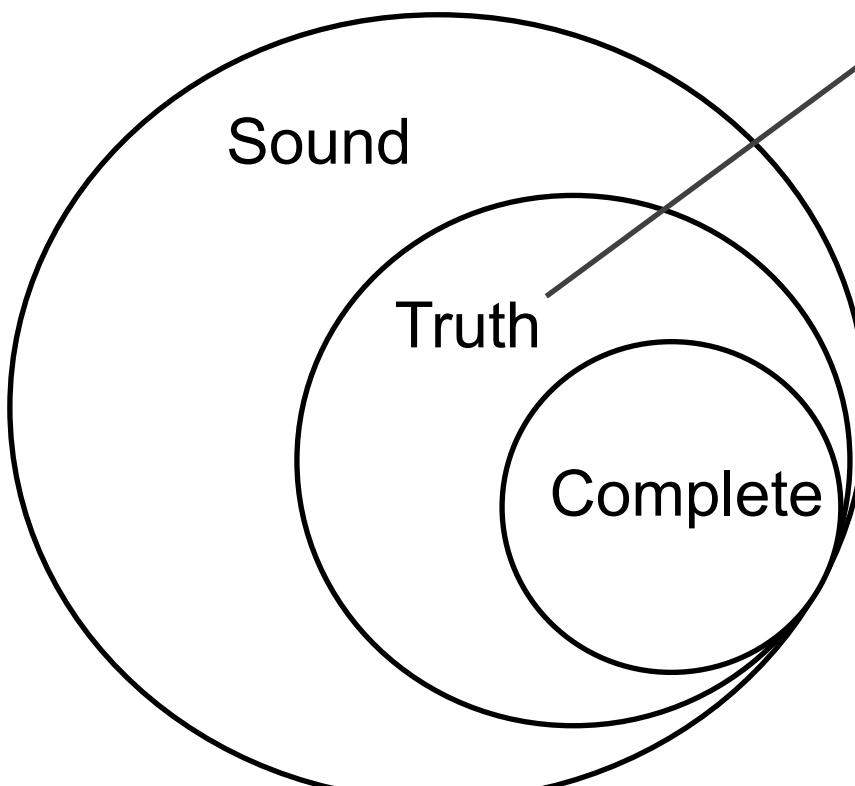
Perfect static analysis

AND

- Sound
- Complete



Rice



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

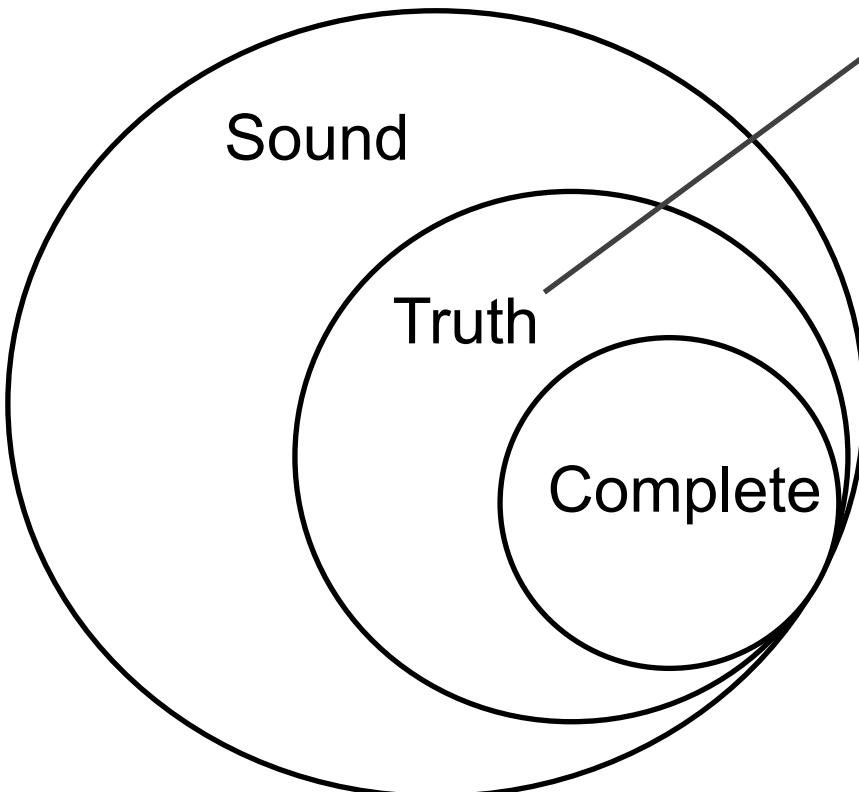
Perfect static analysis

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

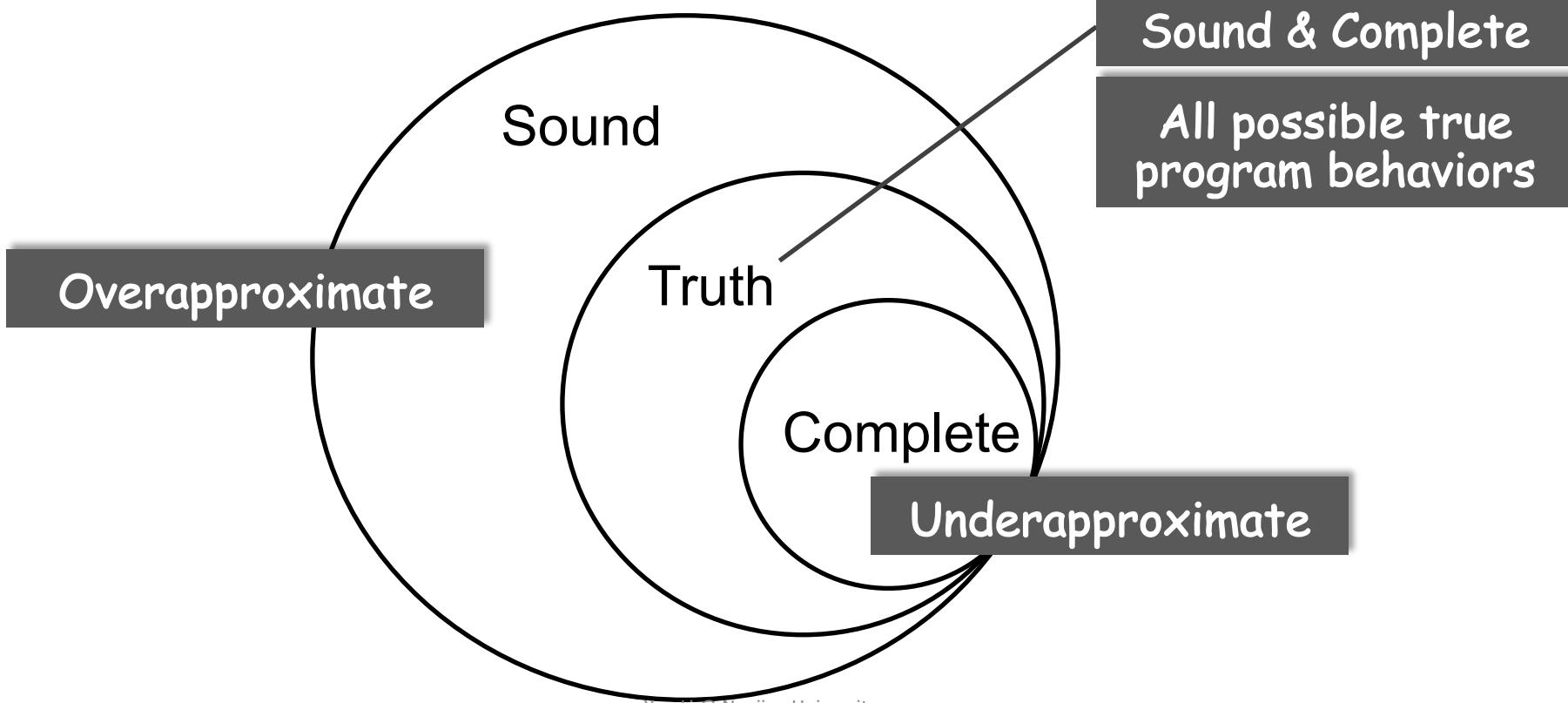
Perfect static analysis

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 program behaviors

NO perfect static analysis!

The end of story ???

Sound

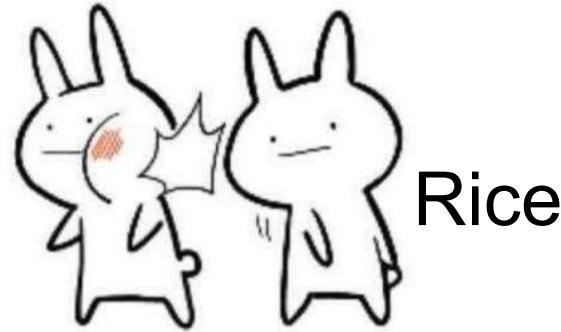
Underapproximate



Perfect static analysis

AND

- Sound
- Complete

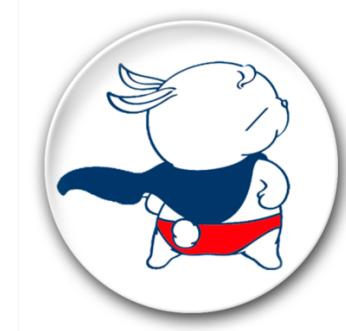


Useful static analysis

OR



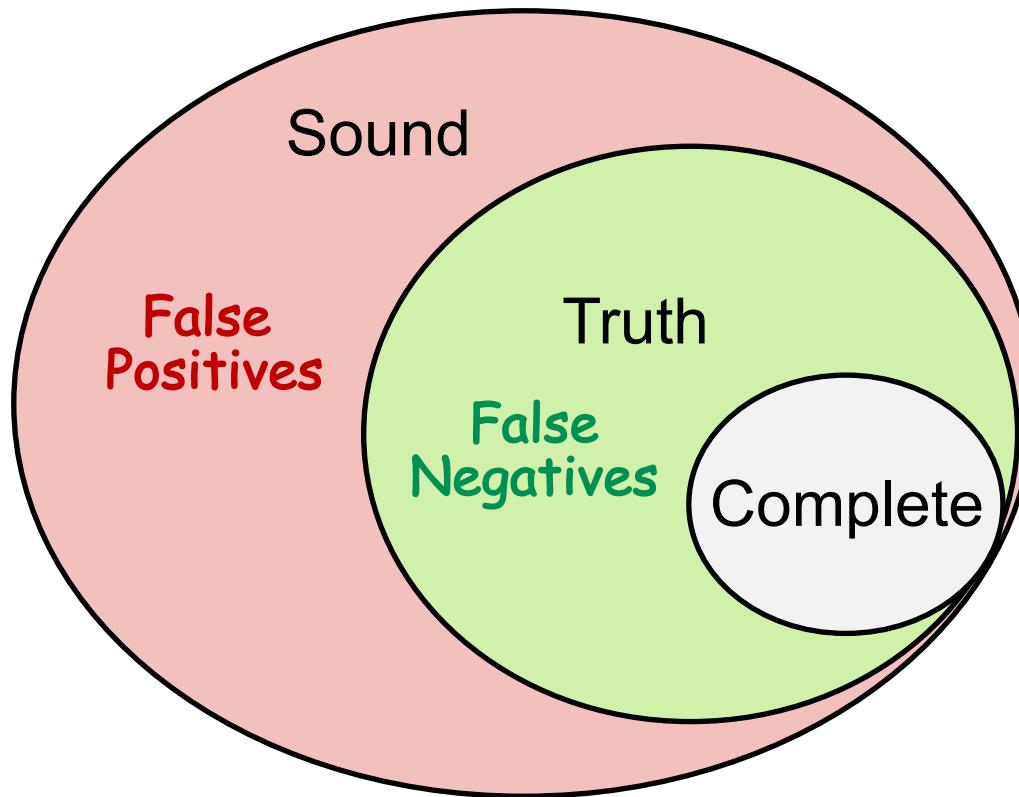
- Compromise soundness (false negatives)
- Compromise completeness (false positives)

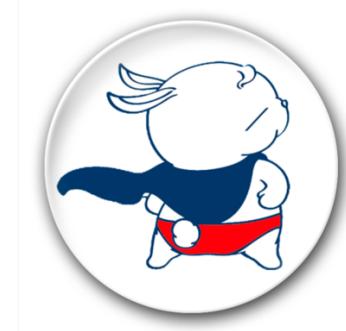


Useful static analysis

OR

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

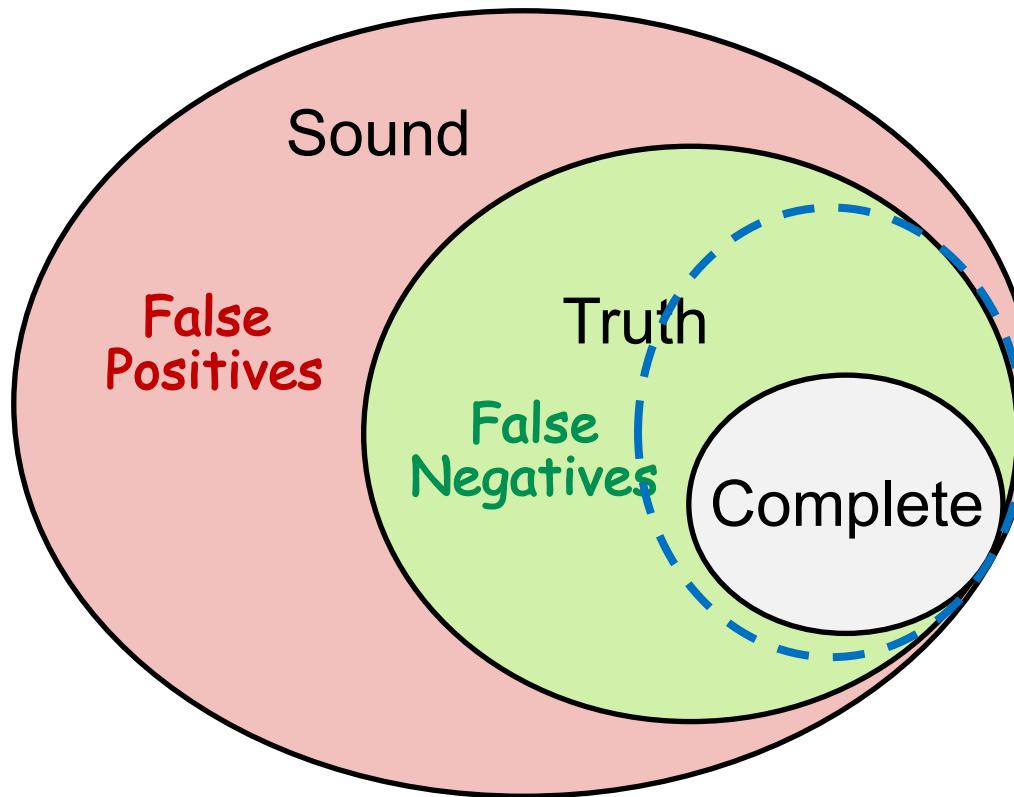




Useful static analysis

OR

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

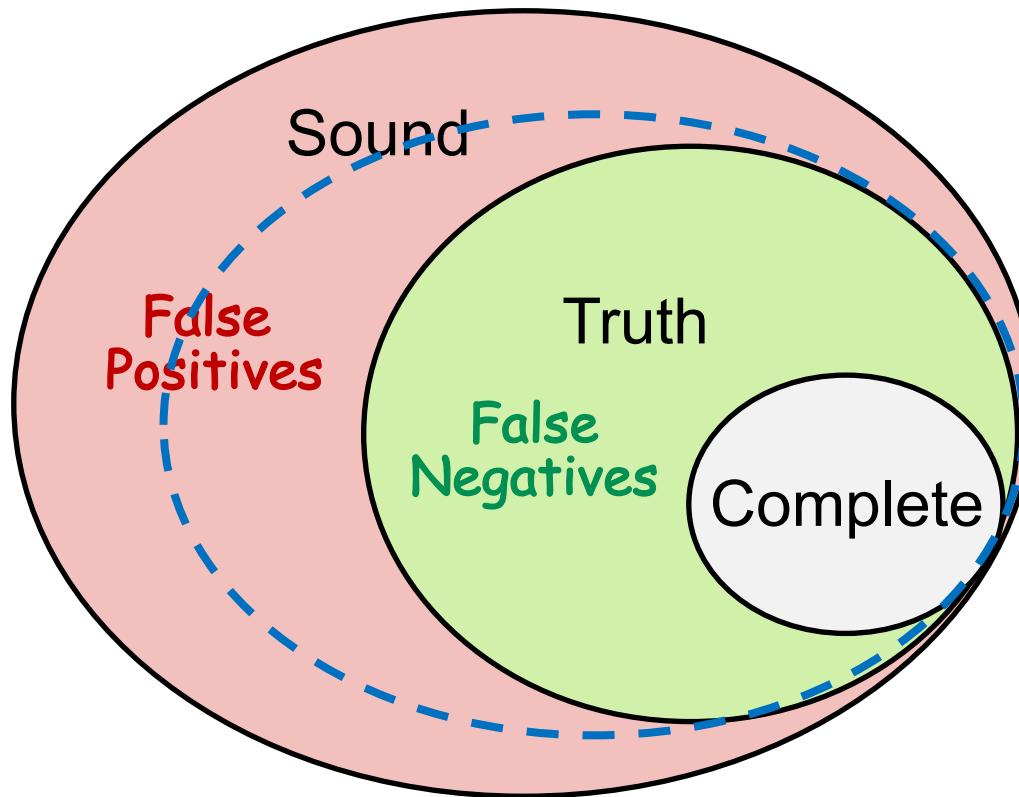




Useful static analysis

OR

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

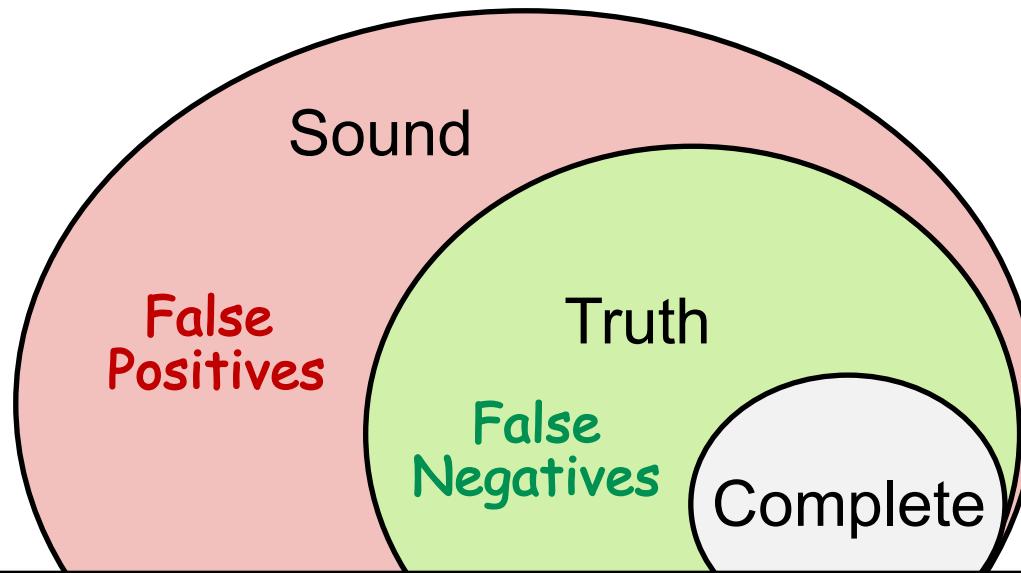


Useful static analysis



OR

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



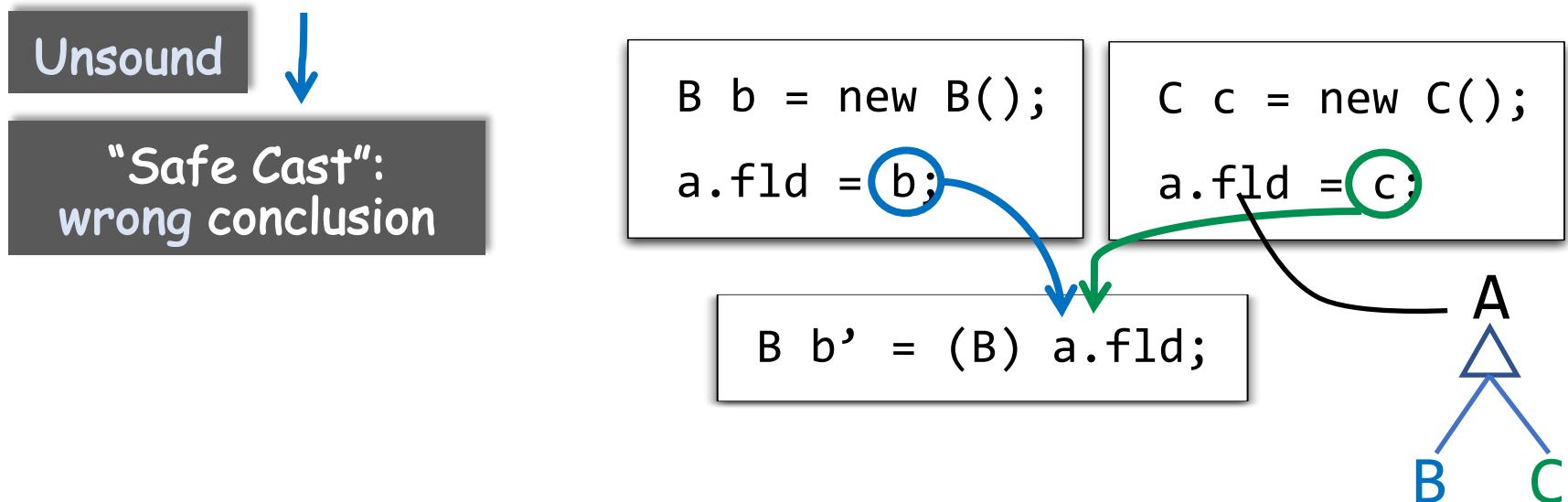
**Mostly compromising completeness:
Sound but not fully-precise static analysis**

Necessity of Soundness

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

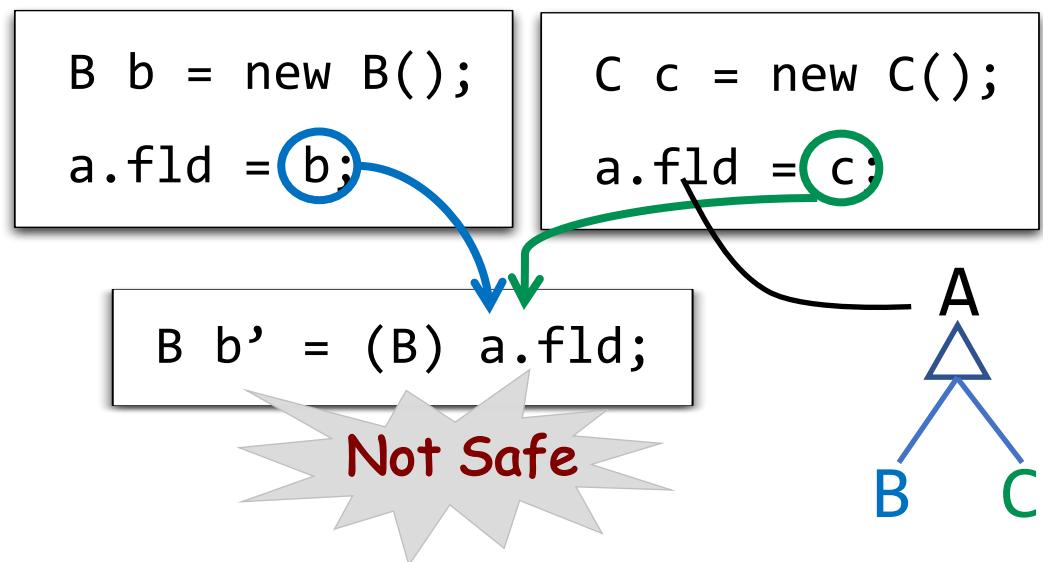
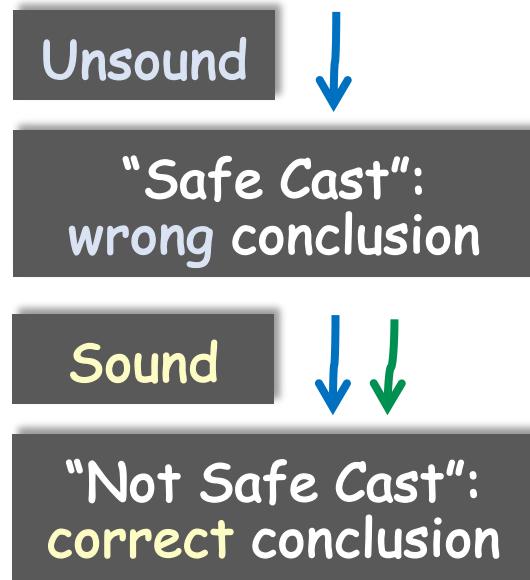
Necessity of Soundness

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



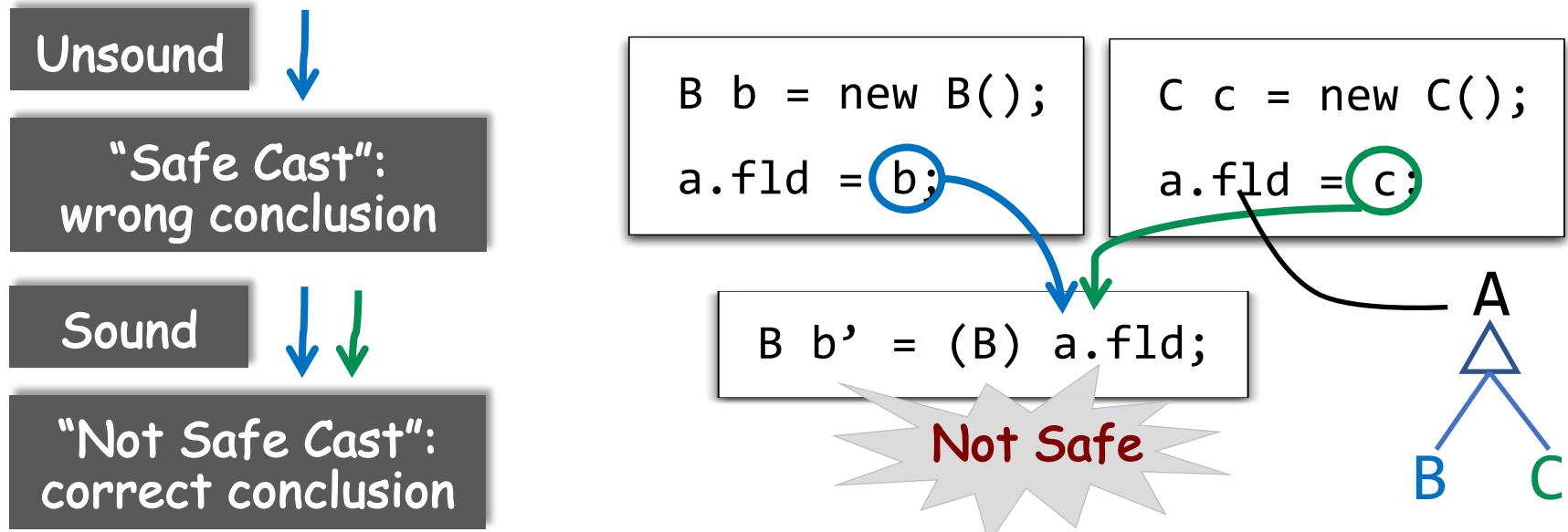
Necessity of Soundness

- 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.

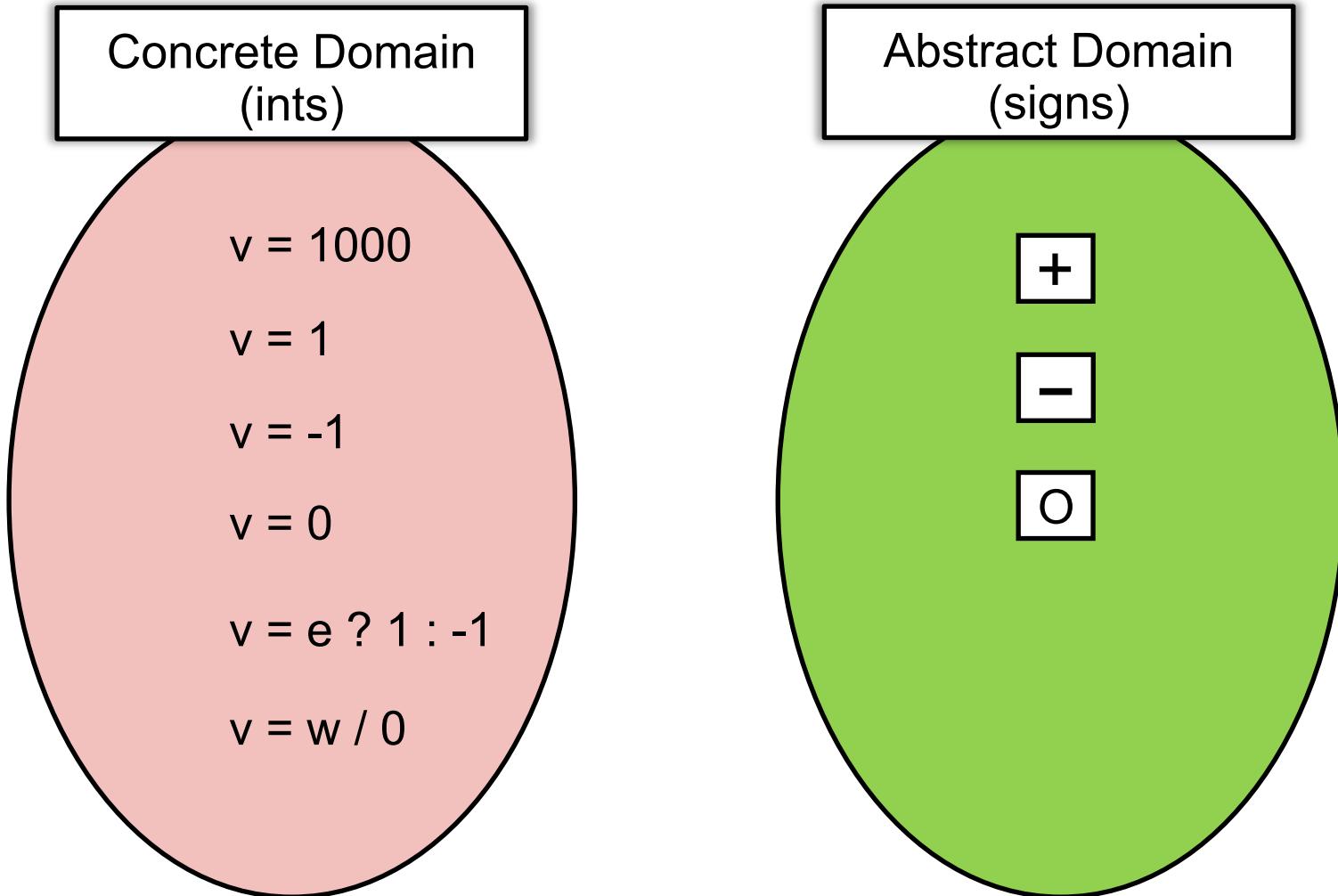
- Abstraction
- Over-approximation
 - Transfer functions
 - Control flows

To check divided
by zero error

To check negative
array indices

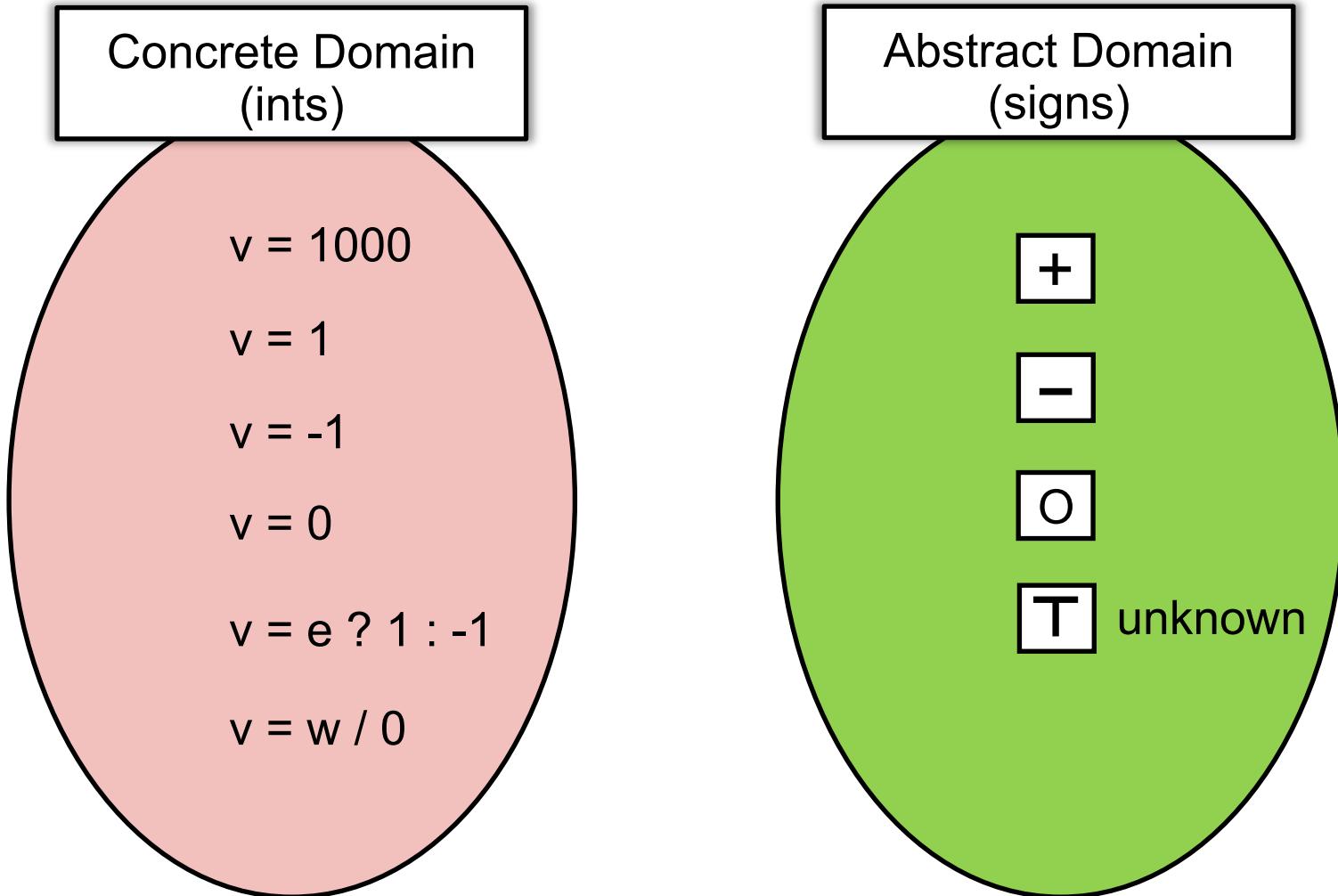
Abstraction

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



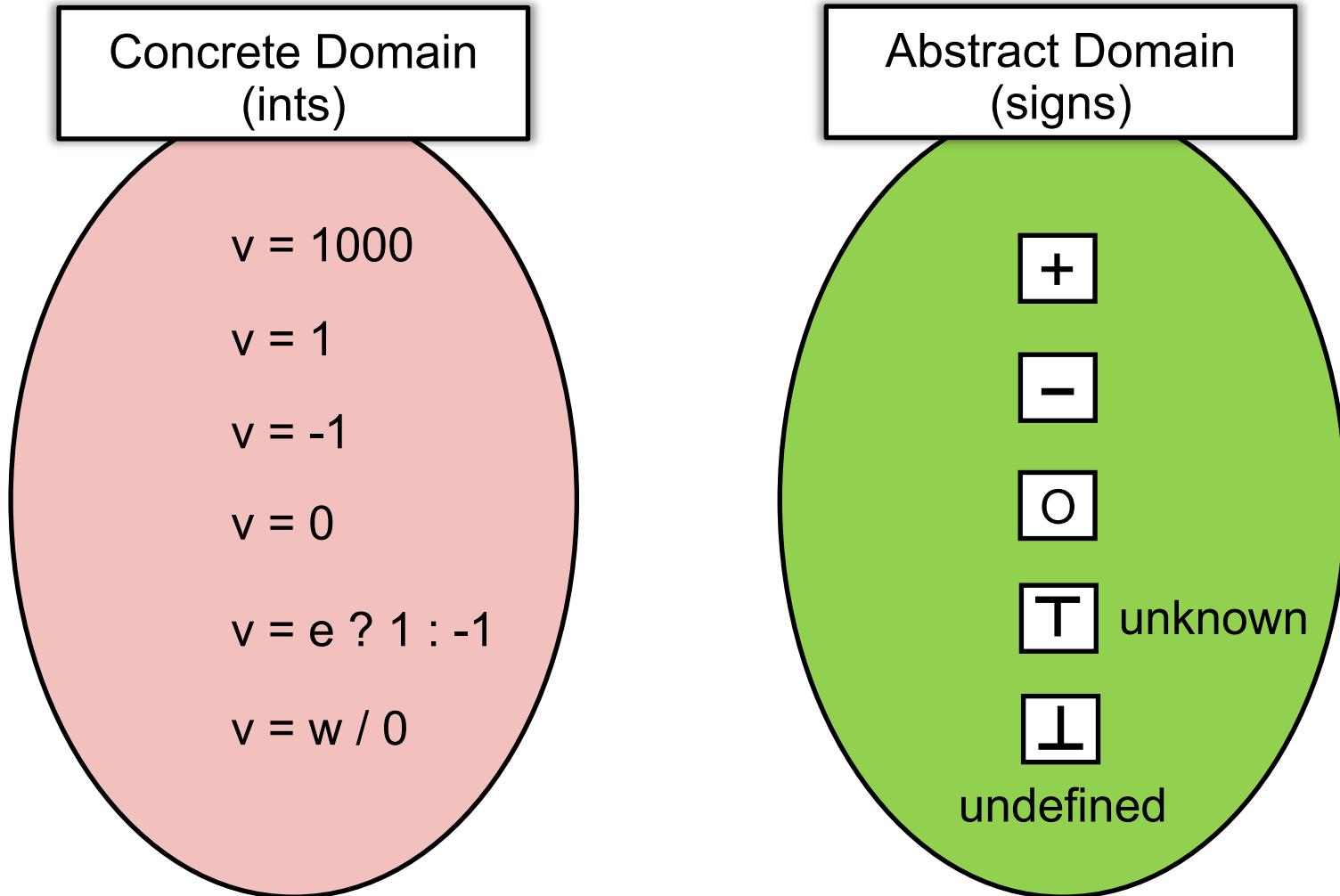
Abstraction

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



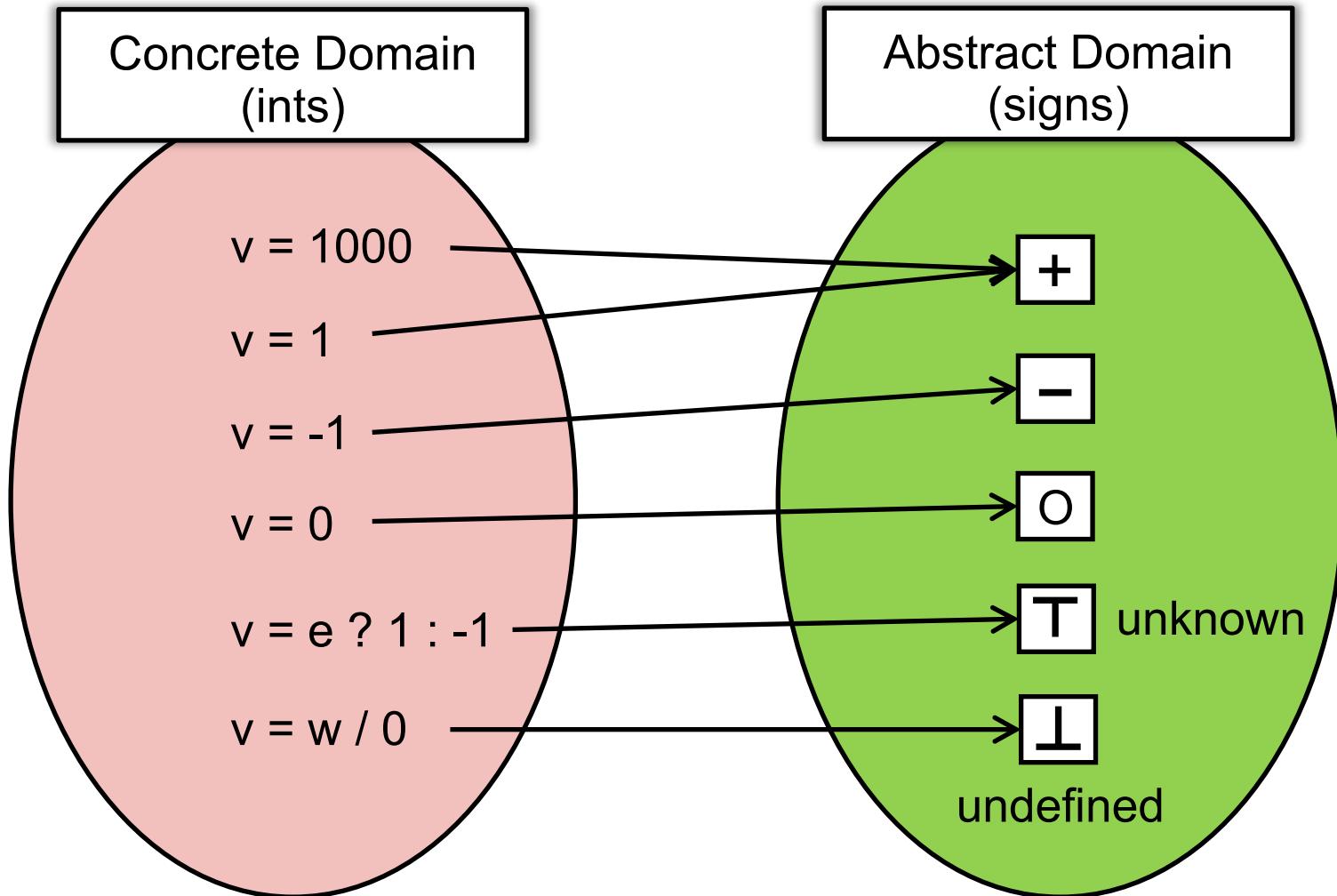
Abstraction

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.

$$\boxed{+} \textcolor{blue}{+} \boxed{+} =$$

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

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

$$\boxed{-} \textcolor{blue}{/} \boxed{-} =$$

$$\boxed{o} \textcolor{blue}{+} \boxed{o} =$$

$$\boxed{T} \textcolor{blue}{/} \boxed{o} =$$

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

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

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.

$$\boxed{+} \textcolor{blue}{+} \boxed{+} = \boxed{+}$$

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

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

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

$$\boxed{o} \textcolor{blue}{+} \boxed{o} = \boxed{o}$$

$$\boxed{T} \textcolor{blue}{/} \boxed{o} =$$

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

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

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.

$$\boxed{+} \textcolor{blue}{+} \boxed{+} = \boxed{+}$$

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

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

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

$$\boxed{o} \textcolor{blue}{+} \boxed{o} = \boxed{o}$$

$$\boxed{T} \textcolor{blue}{/} \boxed{o} =$$

$$\boxed{+} \textcolor{blue}{+} \boxed{-} = \boxed{T}$$

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

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.

$$\boxed{+} \textcolor{blue}{+} \boxed{+} = \boxed{+}$$

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

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

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

$$\boxed{o} \textcolor{blue}{+} \boxed{o} = \boxed{o}$$

$$\boxed{T} \textcolor{blue}{/} \boxed{o} = \boxed{\perp}$$

$$\boxed{+} \textcolor{blue}{+} \boxed{-} = \boxed{T}$$

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

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

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

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

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

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

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

$$\boxed{T} / \boxed{o} = \boxed{\perp}$$

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

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

.....

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

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

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

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

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

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

$$\boxed{T} / \boxed{o} = \boxed{\perp}$$

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

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

....

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

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

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

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

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

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

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

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

$$\boxed{T} / \boxed{O} = \boxed{\perp}$$

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

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

....

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

```
x = 

|   |
|---|
| + |
|---|

  
y = 

|   |
|---|
| - |
|---|

  
z = 

|   |
|---|
| O |
|---|

  
  
a =  
b =  
c =  
p =  
q =
```

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

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

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

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

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

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

$$\boxed{T} / \boxed{O} = \boxed{\perp}$$

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

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

....

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

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

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

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

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

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

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

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

$$\boxed{T} / \boxed{O} = \boxed{\perp}$$

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

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

....

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

```
x = +  
y = -  
z = O  
a = T  
b = O  
c =  
p =  
q =
```

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

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

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

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

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

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

$$\boxed{T} / \boxed{O} = \boxed{\perp}$$

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

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

....

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

```
x = +  
y = -  
z = O  
a = T  
b = O  
c = ⊥  
p =  
q =
```

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

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

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

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

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

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

$$\boxed{T} / \boxed{O} = \boxed{\perp}$$

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

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

....

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

```
x = 

|   |
|---|
| + |
|---|

  
y = 

|   |
|---|
| - |
|---|

  
z = 

|   |
|---|
| O |
|---|

  
a = 

|   |
|---|
| T |
|---|

  
b = 

|   |
|---|
| O |
|---|

  
c = 

|   |
|---|
| ⊥ |
|---|

  
p = 

|   |
|---|
| ⊥ |
|---|

  
q =
```

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

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

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

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

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

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

$$\boxed{T} / \boxed{O} = \boxed{\perp}$$

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

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

....

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

```
x = +  
y = -  
z = O  
a = T  
b = O  
c = ⊥  
p = ⊥  
q = ⊥
```

$$+ + = +$$

$$- - = -$$

$$o o = o$$

$$+ - = \top$$

$$+ / + = +$$

$$- / - = +$$

$$\top / o = \perp$$

$$+ / - = -$$

$$o / - = o$$

....

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

x = +
y = -
z = o
a = \top
b = o
c = \perp
p = \perp
q = \perp

Divided
by zero

$$+ + = +$$

$$- - = -$$

$$o + o = o$$

$$+ - = \perp$$

$$+ / + = +$$

$$- / - = +$$

$$T / o = \perp$$

$$+ / - = -$$

$$o / - = o$$

....

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

- 1
- 2
- 3

x = +
y = -
z = O
a = T
b = O
c = \perp
p = \perp
q = \perp

Divided by zero
negative array index

$$+ + = +$$

$$- - = -$$

$$o + o = o$$

$$+ - = \perp$$

$$+ / + = +$$

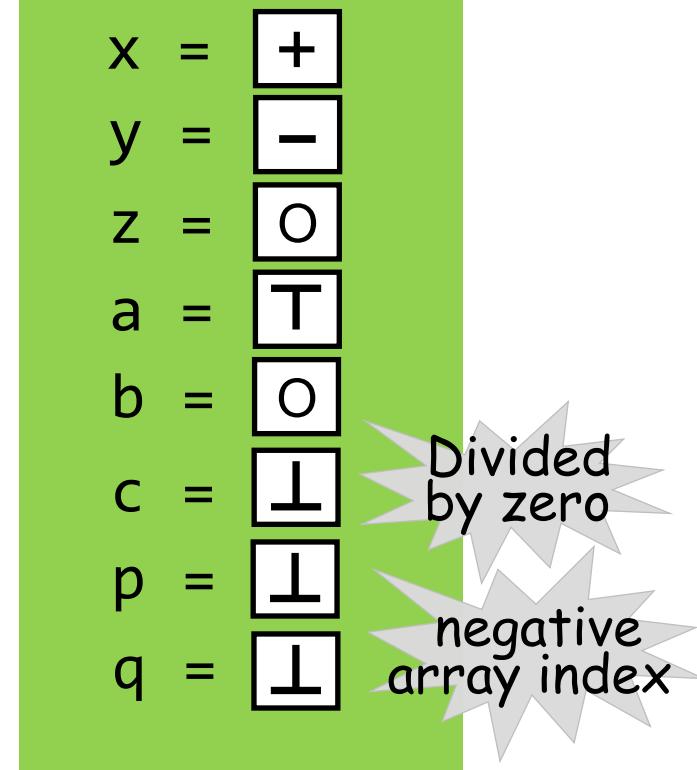
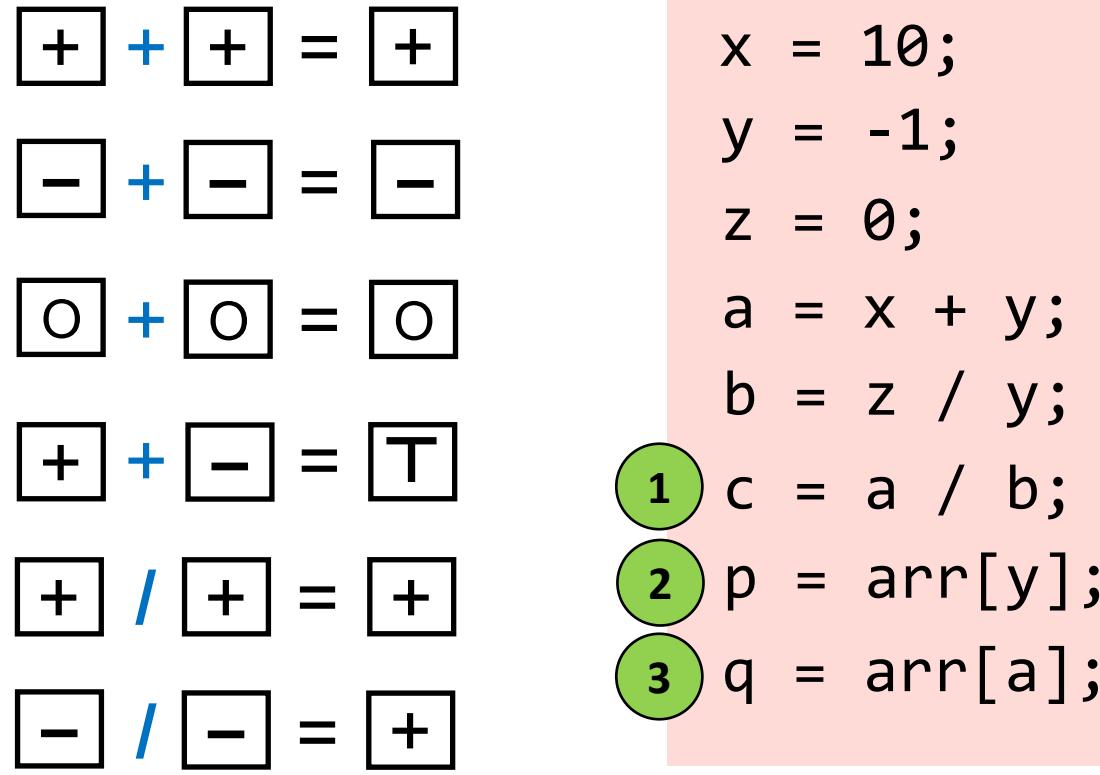
$$- / - = +$$

$$T / o = \perp$$

$$+ / - = -$$

$$o / - = o$$

....



1 2

Static analysis is useful

$$+ + = +$$

$$- - = -$$

$$o + o = o$$

$$+ - = \perp$$

$$+ / + = +$$

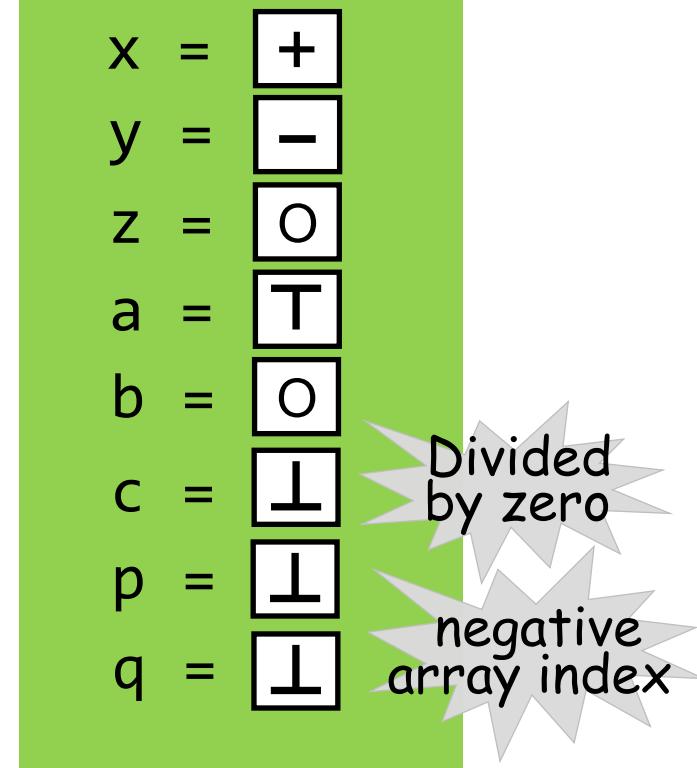
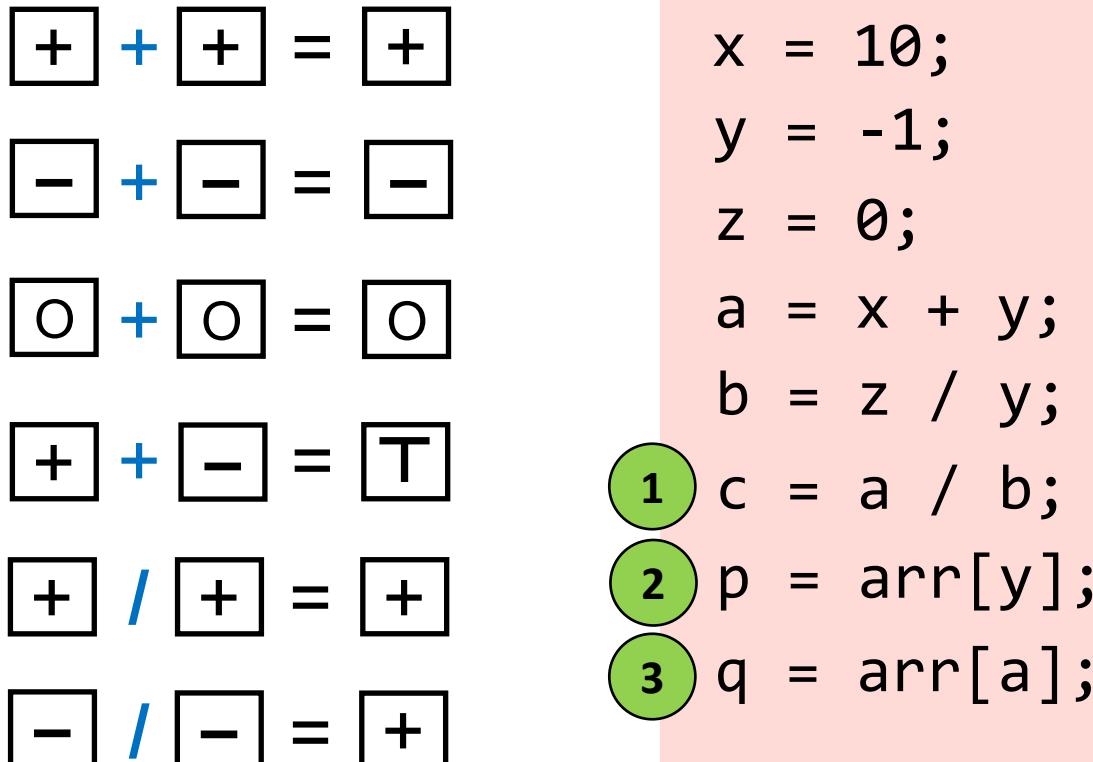
$$- / - = +$$

$$T / o = \perp$$

$$+ / - = -$$

$$o / - = o$$

....

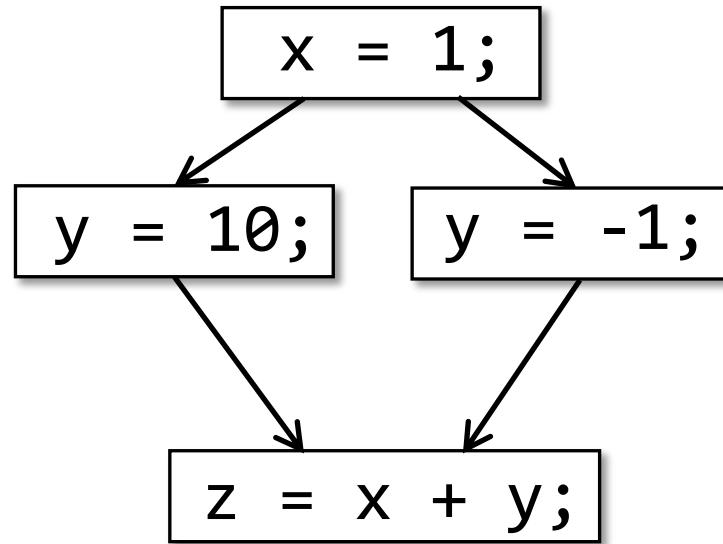


1 2 Static analysis is useful

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

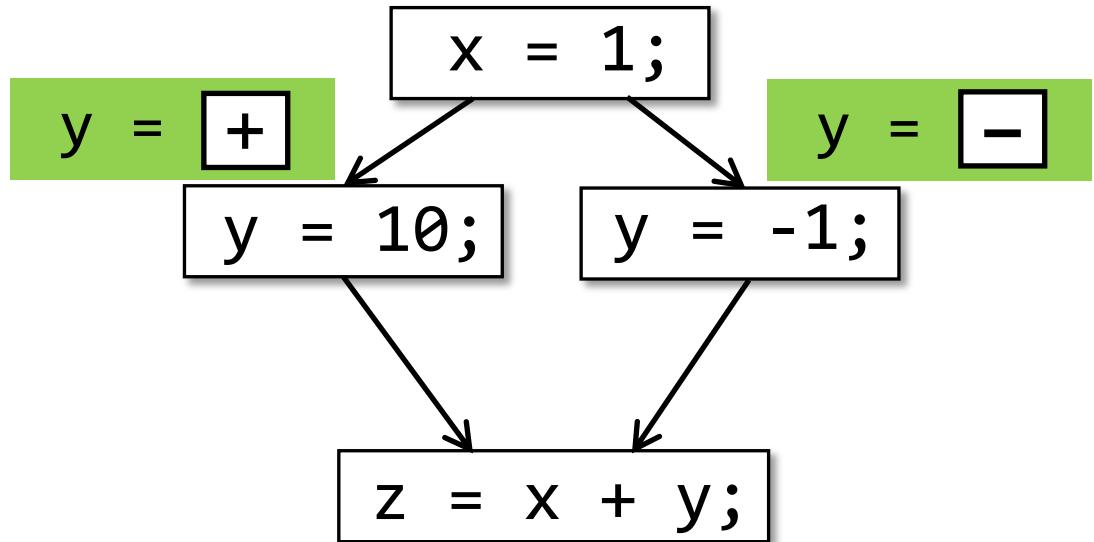
Over-approximation: Control Flows

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



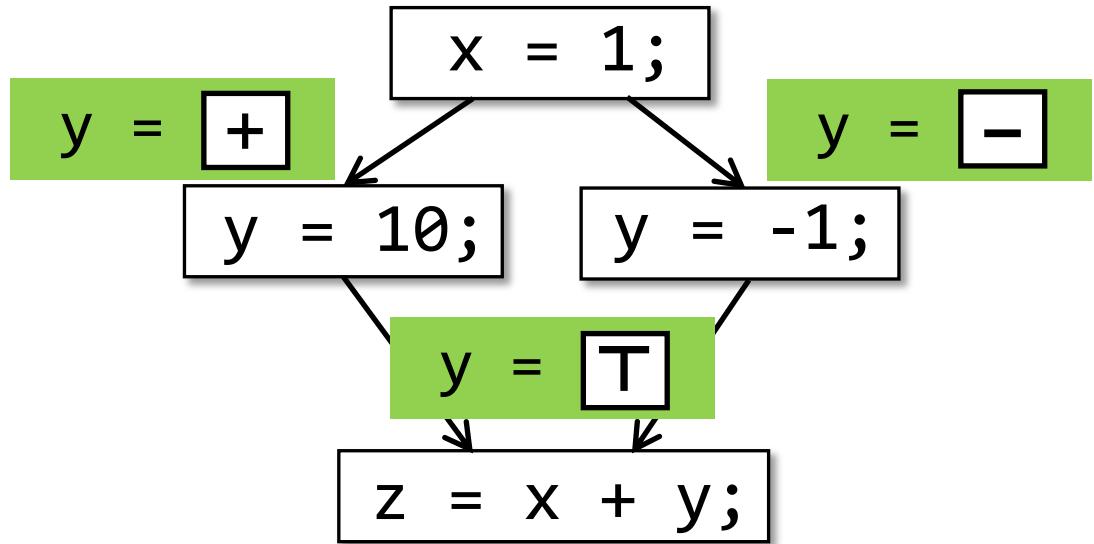
Over-approximation: Control Flows

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



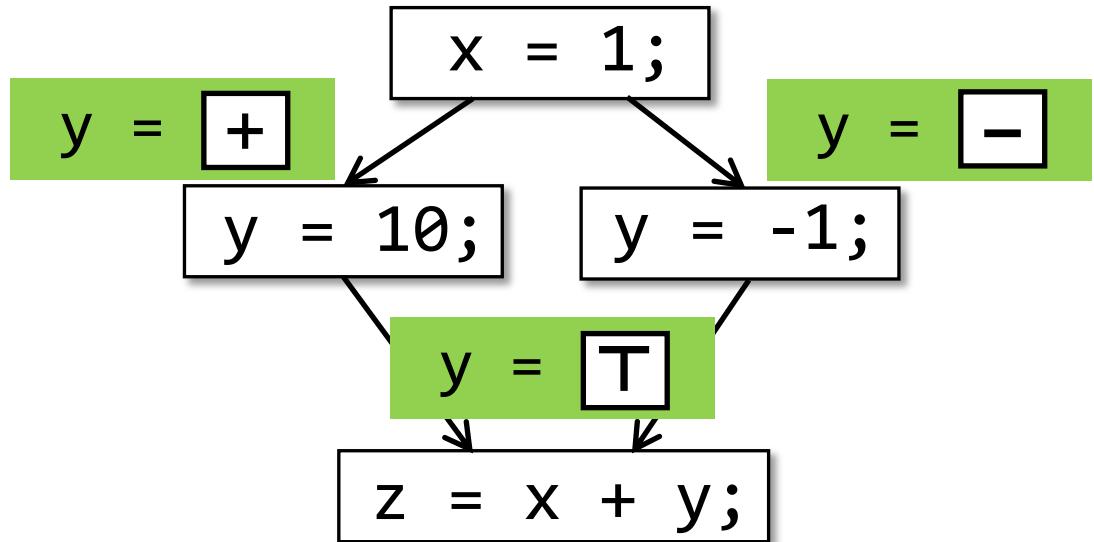
Over-approximation: Control Flows

```
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

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

Evaluation Criteria

- Coding Assignments 40% <http://tai-e.pascal-lab.net>
- Final Exam 60%

OJ注意事项：

- 用学校邮箱注册
- 填写 Student ID
- Nickname 用真实姓名
- 抄袭：第一次发现，该次作业0分
 第二次发现，课程0分
- 不交：得5%诚信分
- 迟交：扣得分 $5\% \times D$, D为迟交天数
- 每帮助找出1个新的有效test case, 实验分数+1

Coding Assignments

A1 → A2 → A3

Learn how to incorporate different analyses to build new analysis

Intraprocedural

A2 → A4 → A7

Learn how to improve analysis precision by handling method calls and aliasing

A5 → A6 → A7/A8

Learn how the precision of fundamental analysis affects the precision of its clients

A1 Live Variable Analysis and Iterative Solver

A2 Constant Propagation and Worklist Solver

A3 Dead Code Detection

A4 CHA and Interprocedural Constant Propagation

A5 Context-Insensitive Pointer/Alias Analysis

A6 Context-Sensitive Pointer/Alias Analysis

A7 Alias-Aware Interprocedural Constant Propagation

A8 Taint Analysis

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?

注意注意！
划重点了！



计算机系楼
副楼536室