Static Program Analysis

Pointer Analysis
Context Sensitivity

Nanjing University

Tian Tan

2020

Problem of Context-Insensitive Pointer Analysis

```
void main() {
  Number n1, n2, x, y;
  n1 = new One(); // o1
  n2 = new Two(); // o2
  x = id(n1);
  y = id(n2);
  int i = x.get();
}
Number id(Number n) {
  return n;
}

interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

Problem of Context-Insensitive Pointer Analysis

```
void main() {
  Number n1, n2, x, y;
  n1 = new One(); // o1
  n2 = new Two(); // o2
  x = id(n1);
  y = id(n2);
  int i = x.get();
}
Number id(Number n) {
  return n;
}

interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

Problem of Context-Insensitive Pointer Analysis

```
void main() {
  Number n1, n2, x, y;
  n1 = new One(); // o1
  n2 = new Two(); // o2
  x = id(n1);
  y = id(n2);
  int i = x.get();
}
Number id(Number n) {
  return n;
}

interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

Constant propagation: i = ?

Problem of Context-Insensitive Pointer Analysis

Context insensitivity, `x.get()`:

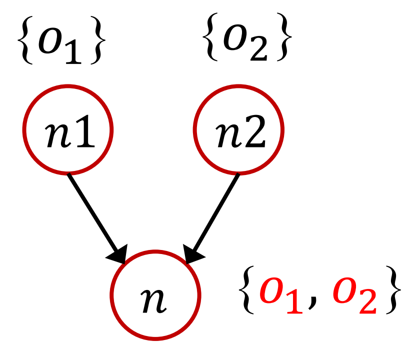
```
void main() {
  Number n1, n2, x, y;
  n1 = new One(); // o1
  n2 = new Two(); // o2
  x = id(n1);
  y = id(n2);
  int i = x.get();
}
Number id(Number n) {
  return n;
}

interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

Problem of Context-Insensitive Pointer Analysis

Context insensitivity, `x.get()`:

```
void main() {
  Number n1, n2, x, y;
  n1 = new One(); // o1
  n2 = new Two(); // o2
  x = id(n1);
  y = id(n2);
  int i = x.get();
}
Number id(Number n) {
  return n;
}

interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

PFG:

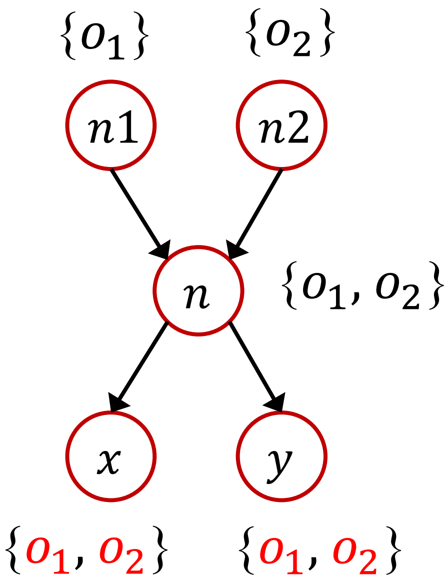


Problem of Context-Insensitive Pointer Analysis

Context insensitivity, `x.get()`:

```
void main() {
  Number n1, n2, x, y;
  n1 = new One(); // o1
  n2 = new Two(); // o2
  x = id(n1);
  y = id(n2);
  int i = x.get();
}
Number id(Number n) {
  return n;
}

interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

PFG:



Problem of Context-Insensitive Pointer Analysis

```
void main() {
  Number n1, n2, x, y;
  n1 = new One(); // o1
  n2 = new Two(); // o2
  x = id(n1);
  y = id(n2);
  int i = x.get();
}
Number id(Number n) {
  return n;
}

interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

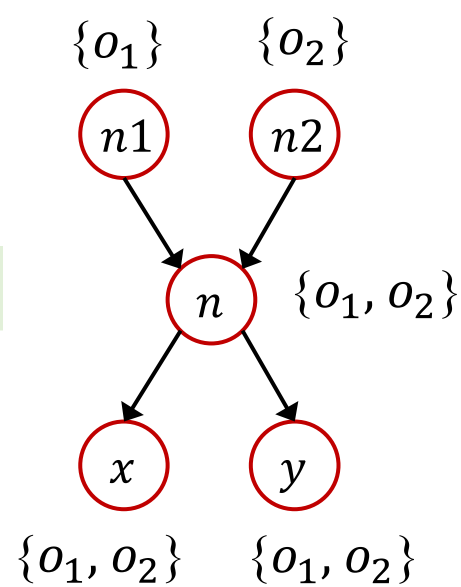
Context insensitivity, `x.get()`:

- 2 call targets

Constant propagation

- `i` = NAC

PFG:



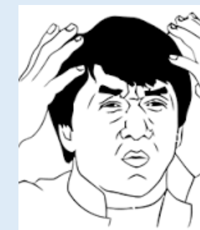
Problem of Context-Insensitive Pointer Analysis

```
void main() {
  Number n1, n2, x, y;
  n1 = new One(); // o1
  n2 = new Two(); // o2
  x = id(n1);
  y = id(n2);
  int i = x.get();
}
Number id(Number n) {
  return n;
}

interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

X

Context insensitivity, `x.get()`:

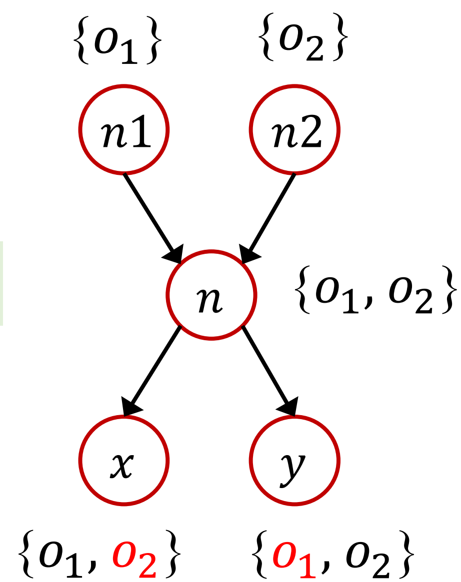
- 2 call targets
- 1 false positive



Constant propagation

- `i` = NAC

PFG:



Via Context-Sensitive Pointer Analysis

Context sensitivity, `x.get()`:

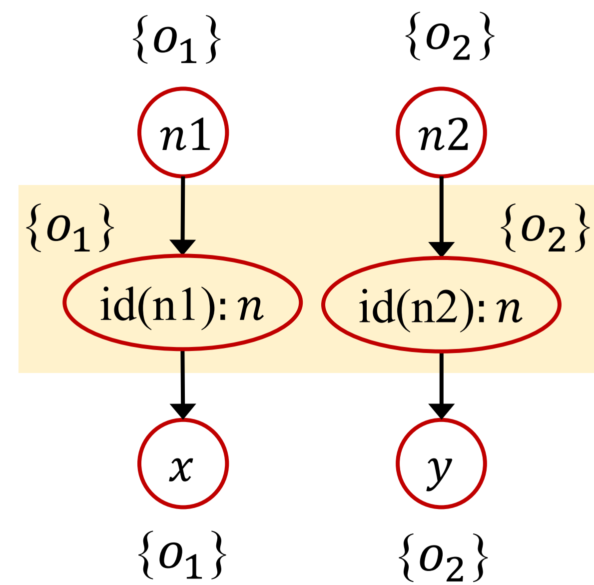
```
void main() {
  Number n1, n2, x, y;
  n1 = new One(); // o1
  n2 = new Two(); // o2
  x = id(n1);
  y = id(n2);
  int i = x.get();
}
Number id(Number n) {
  return n;
}

interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

Via Context-Sensitive Pointer Analysis

Context sensitivity, `x.get()`:

```
void main() {
  Number n1, n2, x, y;
  n1 = new One(); // o1
  n2 = new Two(); // o2
  x = id(n1);
  y = id(n2);
  int i = x.get();
}
Number id(Number n) {
  return n;
}

interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

PFG:



Via Context-Sensitive Pointer Analysis

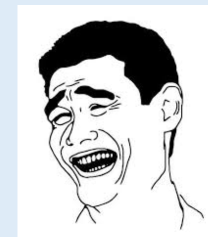
```
void main() {
  Number n1, n2, x, y;
  n1 = new One(); // o1
  n2 = new Two(); // o2
  x = id(n1);
  y = id(n2);
  int i = x.get();
}
Number id(Number n) {
  return n;
}

interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

Context sensitivity, `x.get()`:

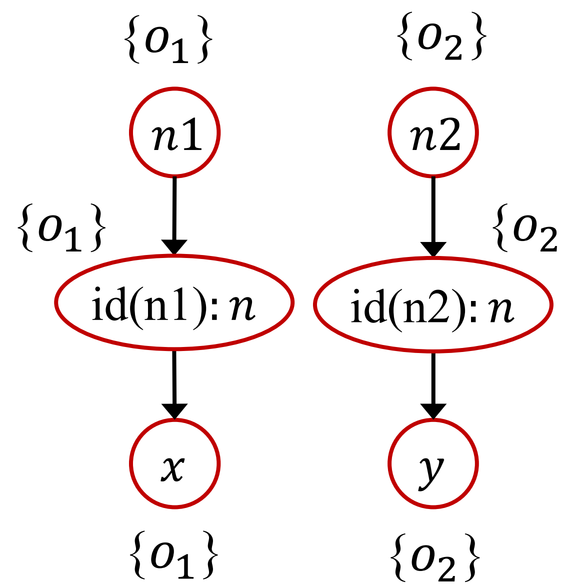
- 1 call targets
- 0 false positive

Constant propagation

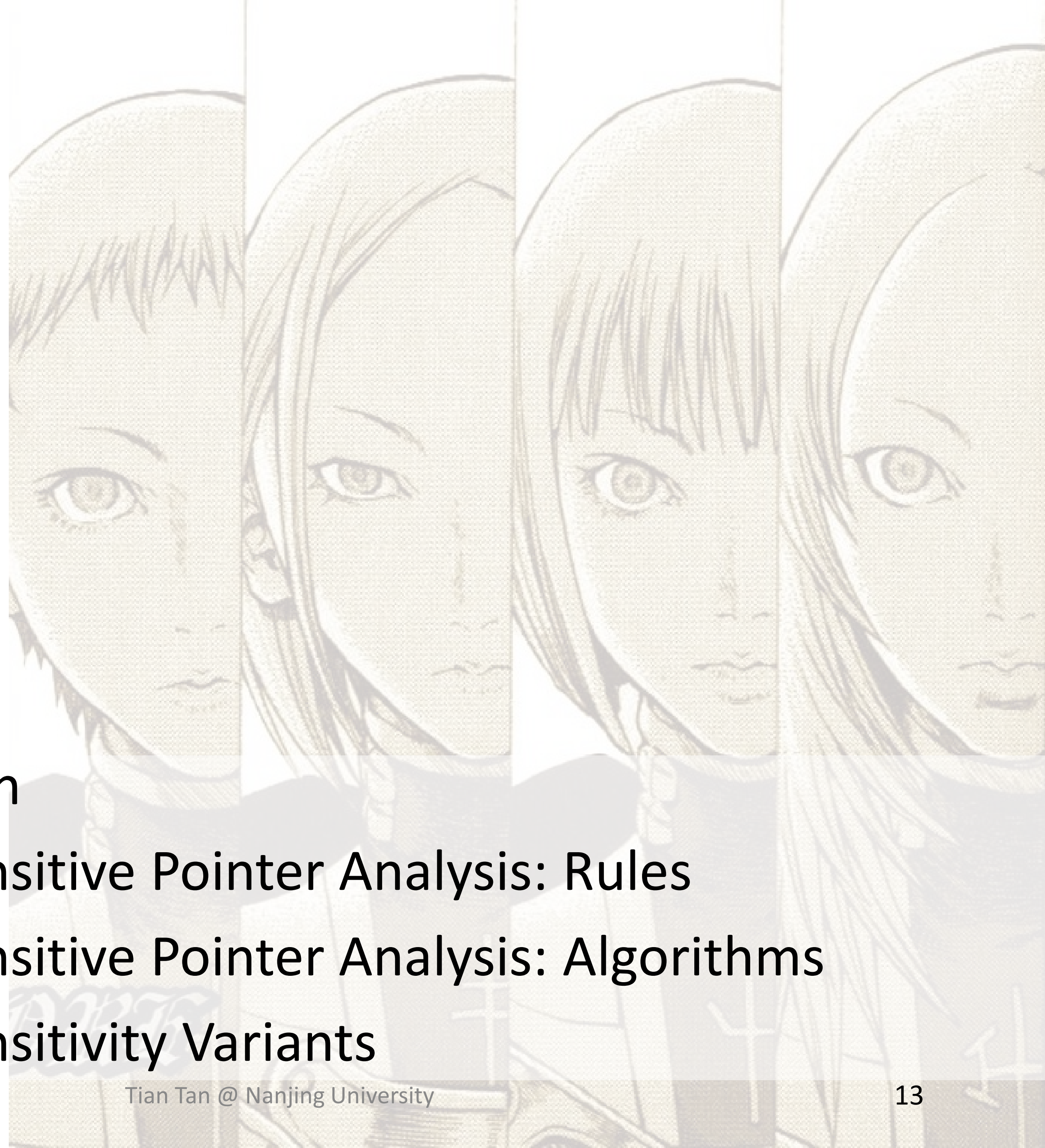
- i = 1



PFG:

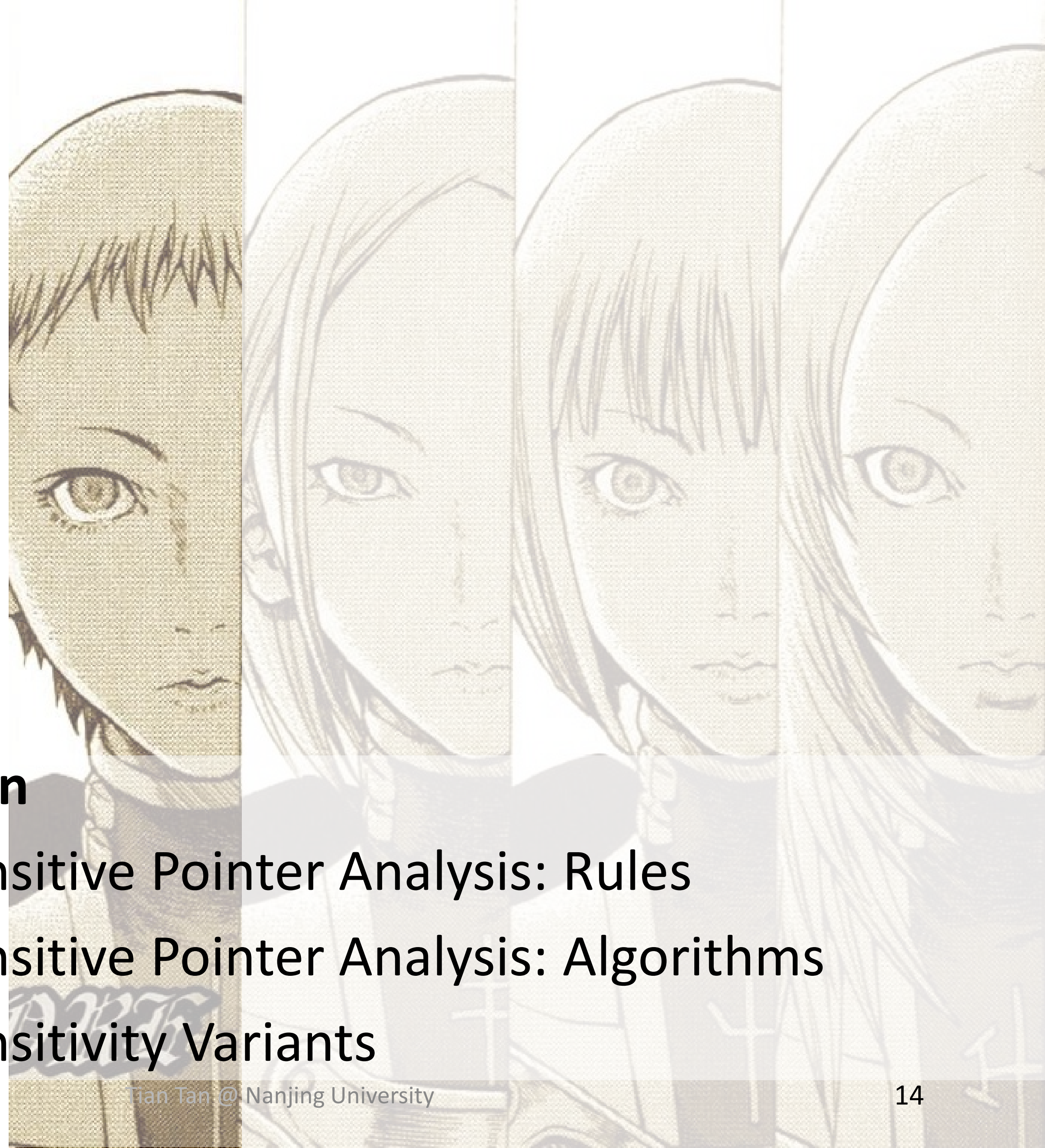


Contents



1. Introduction
2. Context Sensitive Pointer Analysis: Rules
3. Context Sensitive Pointer Analysis: Algorithms
4. Context Sensitivity Variants

Contents



Imprecision of Context Insensitivity (C.I.)

- In dynamic execution, a method may be called multiple times under different calling contexts

```
x = id(n1);
y = id(n2);
int i = x.get();

Number id(Number n) {
  return n;
}
```

Imprecision of Context Insensitivity (C.I.)

- In dynamic execution, a method may be called multiple times under different calling contexts

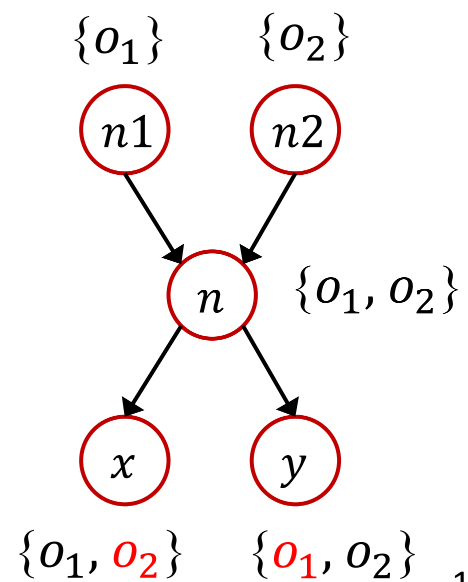
- Under different calling contexts, the variables of the method may point to different objects

```
x = id(n1);
y = id(n2);
int i = x.get();

Number id(Number n) {
  return n;
}
```

Imprecision of Context Insensitivity (C.I.)

- In dynamic execution, a method may be called multiple times under different calling contexts
- Under different calling contexts, the variables of the method may point to different objects
- In C.I. pointer analysis, objects under different contexts are mixed and propagated to other parts of program (through return values or side-effects), causing spurious data flows

```
x = id(n1);
y = id(n2);
int i = x.get();

Number id(Number n) {
  return n;
}
```



Context Sensitivity (C.S.)

- Context sensitivity models calling contexts by distinguishing different data flows of different contexts to improve precision

Context Sensitivity (C.S.)

- Context sensitivity models calling contexts by distinguishing different data flows of different contexts to improve precision

- The oldest and best-known context sensitivity strategy is call-site sensitivity (call-string)
  - Which represents each context of a method as a chain of call sites, i.e., a call site of the method, a call site of the caller, a call site of caller of caller, etc.

  (abstract call stacks in dynamic execution)

Context Sensitivity (C.S.)

- Context sensitivity models calling contexts by distinguishing different data flows of different contexts to improve precision

- The oldest and best-known context sensitivity strategy is call-site sensitivity (call-string)
  - Which represents each context of a method as a chain of call sites, i.e., a call site of the method, a call site of the caller, a call site of caller of caller, etc.

  (abstract call stacks in dynamic execution)

```
x = id(n1);
y = id(n2);
int i = x.get();

Number id(Number n) {
  return n;
}
```

In call-site sensitivity, what are the contexts for method `id(Number)`?

Context Sensitivity (C.S.)

- Context sensitivity models calling contexts by distinguishing different data flows of different contexts to improve precision

- The oldest and best-known context sensitivity strategy is call-site sensitivity (call-string)
  - Which represents each context of a method as a chain of call sites, i.e., a call site of the method, a call site of the caller, a call site of caller of caller, etc.

  (abstract call stacks in dynamic execution)

```
1 x = id(n1);
2 y = id(n2);
3 int i = x.get();
4
5 Number id(Number n) {
6   return n;
7 }
```

In call-site sensitivity, method `id(Number)` has two contexts: [1] and [2]

Context Sensitivity (C.S.)

- Context sensitivity models calling contexts by distinguishing different data flows of different contexts to improve precision

- The oldest and best-known context sensitivity strategy is call-site sensitivity (call-string)
    - Which represents each context of a method as a chain of call sites, i.e., a call site of the method, a call site of the caller, a call site of caller of caller, etc.

```
1 x = id(n1);
2 y = id(n2);
3 int i = x.get();
4
5 Number id(Number n) {
6   return n;
7 }
```

You will see other variants of context sensitivity in the next lecture

In call-site sensitivity, method `id(Number)` has two contexts: [1] and [2]

Cloning-Based Context Sensitivity

The most straightforward approach to implement context sensitivity

- In cloning-based context-sensitive pointer analysis, each method is qualified by one or more contexts

Cloning-Based Context Sensitivity

The most straightforward approach to implement context sensitivity

- In cloning-based context-sensitive pointer analysis, each method is qualified by one or more contexts

- The variables are also qualified by contexts (inherited from the method they are declared in)

Cloning-Based Context Sensitivity

The most straightforward approach to implement context sensitivity

- In cloning-based context-sensitive pointer analysis, each method is qualified by one or more contexts
- The variables are also qualified by contexts (inherited from the method they are declared in)
- Essentially each method and its variables are cloned, one clone per context

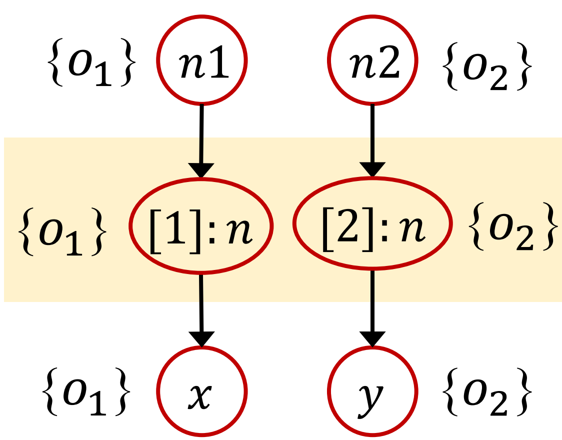
Cloning-Based Context Sensitivity

The most straightforward approach to implement context sensitivity

- In cloning-based context-sensitive pointer analysis, each method is qualified by one or more contexts
- The variables are also qualified by contexts (inherited from the method they are declared in)
- Essentially each method and its variables are cloned, one clone per context

```
1 x = id(n1);
2 y = id(n2);
3 int i = x.get();
4
5 Number id(Number n) {
6   return n;
7 }
```

In call-site sensitivity, method `id(Number)` has two contexts: `[1]` and `[2]`



Context-Sensitive Heap

- OO programs (e.g., Java) are typically heap-intensive
- In practice, to improve precision, context sensitivity should also be applied to heap abstraction
  - The abstract objects are also qualified by contexts (called heap contexts)

Context-Sensitive Heap

- OO programs (e.g., Java) are typically heap-intensive
- In practice, to improve precision, context sensitivity should also be applied to heap abstraction
  - The abstract objects are also qualified by contexts (called heap contexts)

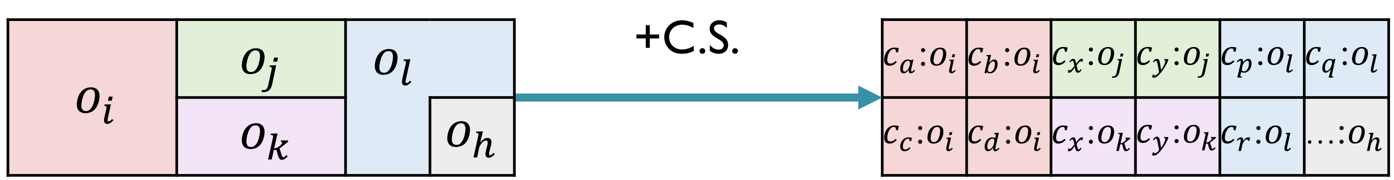
The most common choice is to inherit contexts from the method where the object is allocated

Context-Sensitive Heap

- OO programs (e.g., Java) are typically heap-intensive
- In practice, to improve precision, context sensitivity should also be applied to heap abstraction
  - The abstract objects are also qualified by contexts (called heap contexts)

The most common choice is to inherit contexts from the method where the object is allocated

  - Context-sensitive heap abstraction provides a finer-grained heap model over allocation-site abstraction



Allocation-site abstraction

Context-sensitive allocation-site abstraction

Why Context-Sensitive Heap Improve Precision?

- In dynamic execution, an allocation site can create multiple objects under different calling contexts

```
X newX(…) {
  X x = new X();
  …
  return x;
}
```

Why Context-Sensitive Heap Improve Precision?

- In dynamic execution, an allocation site can create multiple objects under different calling contexts

```
X newX(Y y) {
  X x = new X();
  x.f = y;
  return x;
}
```

- Different objects (allocated by the same site) may be manipulated with different data flows, e.g., stored different values to their fields

Why Context-Sensitive Heap Improve Precision?

- In dynamic execution, an allocation site can create multiple objects under different calling contexts

```
X newX(Y y) {
  X x = new X();
  x.f = y;
  return x;
}
```

- Different objects (allocated by the same site) may be manipulated with different data flows, e.g., stored different values to their fields

> In pointer analysis, analyzing such code without heap contexts may lose precision by merging the data flows of different contexts to one abstract object

> In contrast, distinguishing different objects from the same allocation site by heap contexts gains precision

Context-Sensitive Heap: Example

```
n1 = new One();
n2 = new Two();
x1 = newX(n1);
x2 = newX(n2);
n = x1.f;

X newX(Number p) {
  X x = new X();
  x.f = p;
  return x;
}
class X {
   Number f;
}
```

Context-Sensitive Heap: Example

```
n1 = new One();
n2 = new Two();
x1 = newX(n1);
x2 = newX(n2);
n = x1.f;

X newX(Number p) {
  X x = new X();
  x.f = p;
  return x;
}
class X {
   Number f;
}
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1

?

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11 }
12 class X {
13     Number f;
14 }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8

Context-Sensitive Heap: Example

```
n1 = new One();
n2 = new Two();
x1 = newX(n1);
x2 = newX(n2);
n = x1.f;

X newX(Number p) {
  X x = new X();
  x.f = p;
  return x;
}
class X {
   Number f;
}
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
Field	**Object**
o_8.f	o_1

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
Field	**Object**
o_8.f	o_1

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13    Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
Field	**Object**
o_8.f	o_1

?

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
Field	**Object**
o_8.f	o_1, o_2

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	?

Field	Object
o_8.f	o_1, o_2

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2

Field	Object
o_8.f	o_1, o_2

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2
Field	**Object**
o_8.f	o_1, o_2

Spurious data flow due to lack of C.S. heap

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11 }
12 class X {
13    Number f;
14 }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2
Field	**Object**
o_8.f	o_1, o_2

C.S. + C.S heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1

Spurious data flow due to lack of C.S. heap

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13    Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2
Field	**Object**
o_8.f	o_1, o_2

Spurious data flow due to lack of C.S. heap

C.S. + C.S heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1

?

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13    Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2

Field	Object
o_8.f	o_1, o_2

Spurious data flow due to lack of C.S. heap

C.S. + C.S heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	3:o_8

Object with heap context

Context-Sensitive Heap: Example

```
1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8    X x = new X();
9    x.f = p;
10   return x;
11 }
12 class X {
13    Number f;
14 }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2
Field	**Object**
o_8.f	o_1, o_2

Spurious data flow due to lack of C.S. heap

C.S. + C.S heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	3:o_8

?

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13    Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2
Field	**Object**
o_8.f	o_1, o_2

Spurious data flow due to lack of C.S. heap

C.S. + C.S heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	3:o_8
x1	3:o_8
Field	**Object**
3:o_8.f	o_1

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13    Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2

Field	Object
o_8.f	o_1, o_2

Spurious data flow due to lack of C.S. heap

C.S. + C.S heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	3:o_8
x1	3:o_8
4:p	o_2
4:x	4:o_8

Field	Object
3:o_8.f	o_1

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2
Field	**Object**
o_8.f	o_1, o_2

Spurious data flow due to lack of C.S. heap

C.S. + C.S heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	3:o_8
x1	3:o_8
4:p	o_2
4:x	4:o_8
x2	4:o_8
Field	**Object**
3:o_8.f	o_1
4:o_8.f	o_2

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2

Field	Object
o_8.f	o_1, o_2

Spurious data flow
due to lack of C.S. heap

C.S. + C.S heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	3:o_8
x1	3:o_8
4:p	o_2
4:x	4:o_8
x2	4:o_8
n	?

Field	Object
3:o_8.f	o_1
4:o_8.f	o_2

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2
Field	**Object**
o_8.f	o_1, o_2

Spurious data flow due to lack of C.S. heap

C.S. + C.S heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	3:o_8
x1	3:o_8
4:p	o_2
4:x	4:o_8
x2	4:o_8
n	o_1
Field	**Object**
3:o_8.f	o_1
4:o_8.f	o_2

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.S., no C.S. heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	o_8
x1	o_8
4:p	o_2
4:x	o_8
x2	o_8
n	o_1, o_2
Field	**Object**
o_8.f	o_1, o_2

Spurious data flow due to lack of C.S. heap

C.S. + C.S heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	3:o_8
x1	3:o_8
4:p	o_2
4:x	4:o_8
x2	4:o_8
n	o_1
Field	**Object**
3:o_8.f	o_1
4:o_8.f	o_2

Context-sensitive heap improves precision

Context-Sensitive Heap: Example

```
1   n1 = new One();
2   n2 = new Two();
3   x1 = newX(n1);
4   x2 = newX(n2);
5   n = x1.f;
6
7   X newX(Number p) {
8     X x = new X();
9     x.f = p;
10    return x;
11  }
12  class X {
13     Number f;
14  }
```

C.I. + C.S. heap

Variable	Object
n1	o_1
n2	o_2
p	o_1, o_2
x	$3{:}o_8, 4{:}o_8$
x1	$3{:}o_8, 4{:}o_8$
x2	$3{:}o_8, 4{:}o_8$
n	o_1, o_2
Field	**Object**
$3{:}o_8$.f	o_1, o_2
$4{:}o_8$.f	o_1, o_2

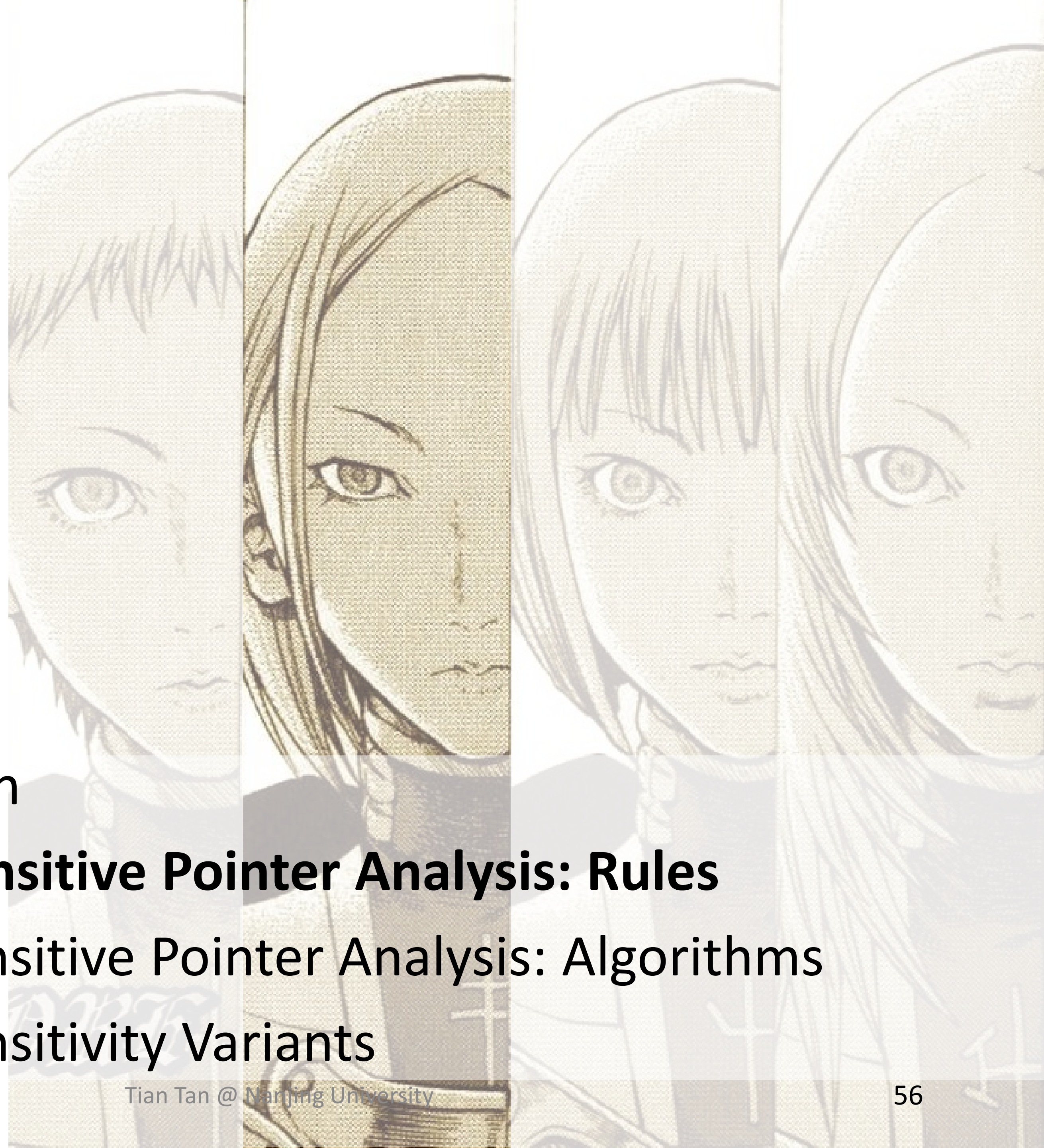
Without C.S., C.S. heap cannot improve precision

C.S. + C.S heap

Variable	Object
n1	o_1
n2	o_2
3:p	o_1
3:x	$3{:}o_8$
x1	$3{:}o_8$
4:p	o_2
4:x	$4{:}o_8$
x2	$4{:}o_8$
n	o_1
Field	**Object**
$3{:}o_8$.f	o_1
$4{:}o_8$.f	o_2

Context-sensitive heap improves precision

Contents



1. Introduction
2. **Context Sensitive Pointer Analysis: Rules**
3. Context Sensitive Pointer Analysis: Algorithms
4. Context Sensitivity Variants

Domain and Notations

In context-sensitive analysis,
program elements are qualified by **contexts**

Context:	c, c', c''	$\in \mathrm{C}$
Context-sensitive methods:	$c: m$	$\in \mathrm{C} \times \mathrm{M}$
Context-sensitive variables:	$c: x, c': y$	$\in \mathrm{C} \times \mathrm{V}$
Context-sensitive objects:	$c: o_i, c': o_j$	$\in \mathrm{C} \times \mathrm{O}$
Fields:	f, g	$\in \mathrm{F}$
Instance fields:	$c: o_i.f, c': o_j.g$	$\in \mathrm{C} \times \mathrm{O} \times \mathrm{F}$
Context-sensitive pointers:	$\mathrm{CSPointer} =$	$(\mathrm{C} \times \mathrm{V}) \cup (\mathrm{C} \times \mathrm{O} \times \mathrm{F})$
Points-to relations:	$pt:$	$\mathrm{CSPointer} \to \mathcal{P}(\mathrm{C} \times \mathrm{O})$

Rules

Kind	Statement	Rule (under context c)
New	`i: x = new T()`	$\overline{c: o_i \in pt(c: x)}$
Assign	`x = y`	$\dfrac{c': o_i \in pt(c: y)}{c': o_i \in pt(c: x)}$
Store	`x.f = y`	$\dfrac{c': o_i \in pt(c: x), c'': o_j \in pt(c: y)}{c'': o_j \in pt(c': o_i.f)}$
Load	`y = x.f`	$\dfrac{c': o_i \in pt(c: x), c'': o_j \in pt(c': o_i.f)}{c'': o_j \in pt(c: y)}$

Rules

Kind	Statement	Rule
New	`i: x = new T()`	$\overline{o_i \in pt(x)}$
Assign	`x = y`	$\dfrac{o_i \in pt(y)}{o_i \in pt(x)}$
Store	`x.f = y`	$\dfrac{o_i \in pt(x),\ o_j \in pt(y)}{o_j \in pt(o_i.f)}$
Load	`y = x.f`	$\dfrac{o_i \in pt(x),\ o_j \in pt(o_i.f)}{o_j \in pt(y)}$

Rules

Kind	Statement	Rule (under context c)
New	`i: x = new T()`	$\overline{c: o_i \in pt(c: x)}$
Assign	`x = y`	$\dfrac{c': o_i \in pt(c: y)}{c': o_i \in pt(c: x)}$
Store	`x.f = y`	$\dfrac{c': o_i \in pt(c: x),\ c'': o_j \in pt(c: y)}{c'': o_j \in pt(c': o_i.f)}$
Load	`y = x.f`	$\dfrac{c': o_i \in pt(c: x),\ c'': o_j \in pt(c': o_i.f)}{c'': o_j \in pt(c: y)}$

Rule: New

$$\frac{}{c{:}\,o_i \in pt(c{:}\,x)}$$

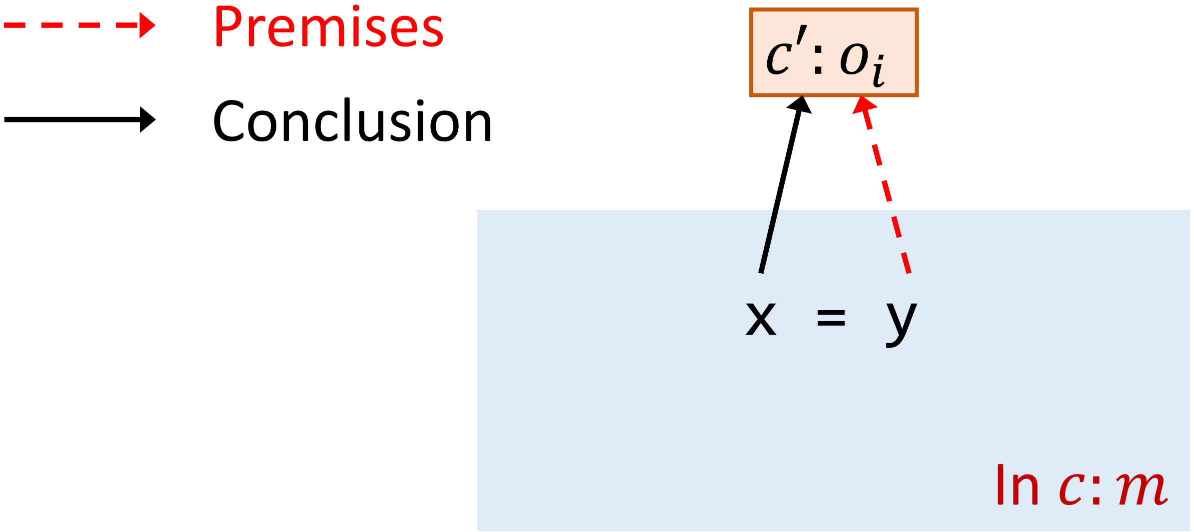
⟶ Conclusion $\boxed{c{:}\,o_i}$

```
i: x = new T()
```

In $c{:}\,m$

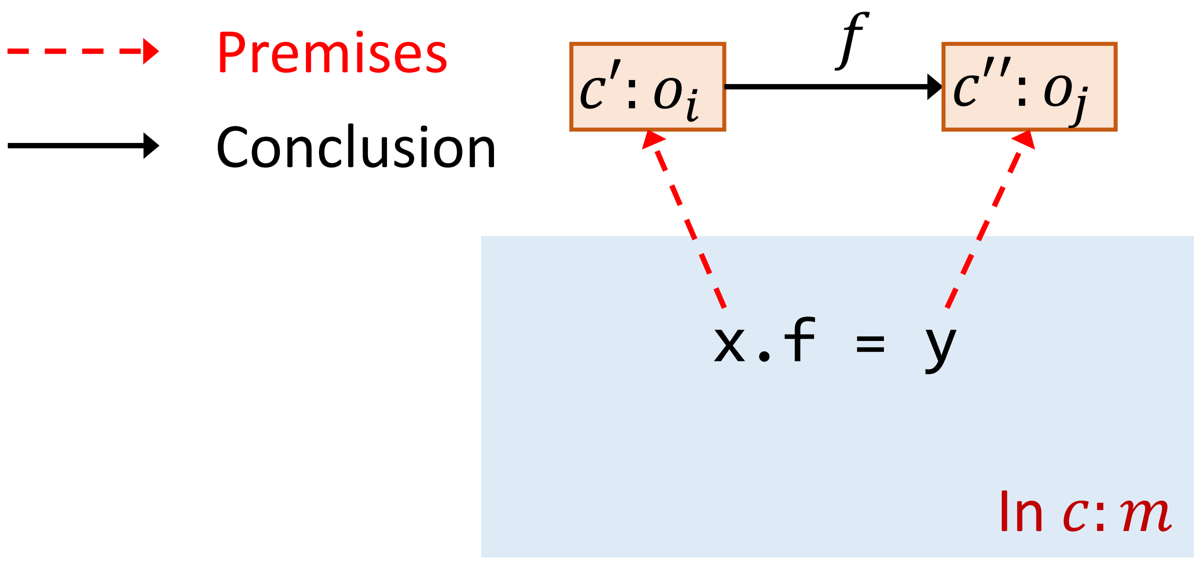
Rule: Assign

$$\frac{c' : o_i \in pt(c : y)}{c' : o_i \in pt(c : x)}$$



Rule: Store

$$\frac{c' : o_i \in pt(c : x),\ c'' : o_j \in pt(c : y)}{c'' : o_j \in pt(c' : o_i.f)}$$

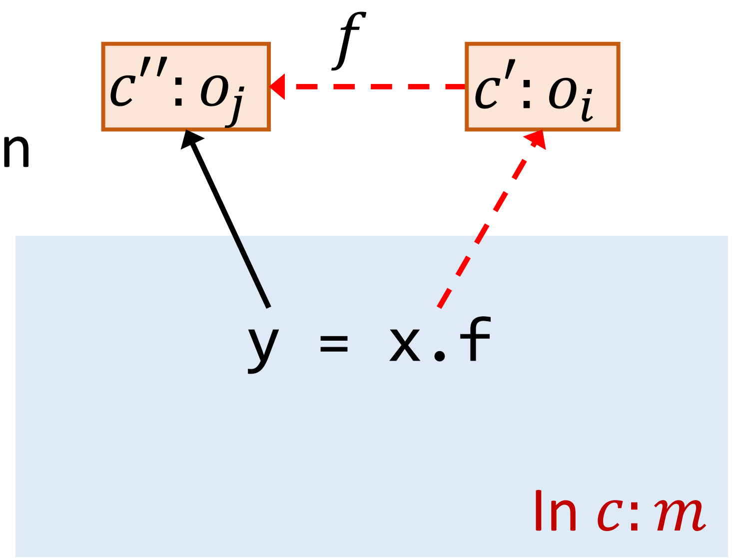


Rule: Load

$$\frac{c':o_i \in pt(c:x),\ c'':o_j \in pt(c':o_i.f)}{c'':o_j \in pt(c:y)}$$

Premises

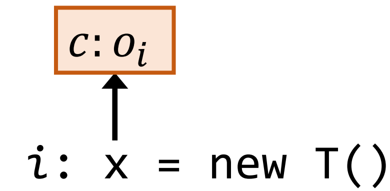
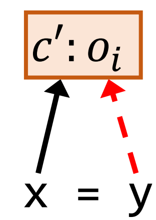
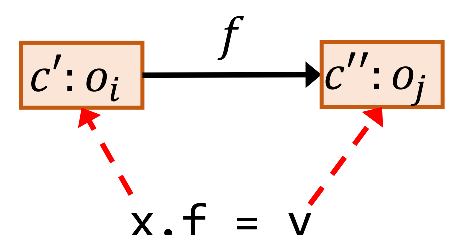
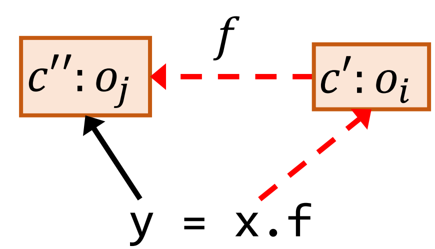
Conclusion



Rules

Premises

Conclusion

Kind	Rule	Illustration
New	$\dfrac{}{c: o_i \in pt(c: x)}$	
Assign	$\dfrac{c': o_i \in pt(c: y)}{c': o_i \in pt(c: x)}$	
Store	$\dfrac{c': o_i \in pt(c: x),\ c'': o_j \in pt(c: y)}{c'': o_j \in pt(c': o_i.f)}$	
Load	$\dfrac{c': o_i \in pt(c: x),\ c'': o_j \in pt(c': o_i.f)}{c'': o_j \in pt(c: y)}$	

Caller context: c

Rule: Call

Kind	Statement	Rule (under context c)
Call	`l: r = x.k(a1,…,an)`	$$\frac{\begin{array}{c} c': o_i \in pt(c{:}x), \\ m = \text{Dispatch}(o_i, \text{k}), c^t = \text{Select}(c, l, c': o_i) \\ c'': o_u \in pt(c{:}aj), 1 \le j \le n \\ c''': o_v \in pt(c^t{:}m_{ret}) \end{array}}{\begin{array}{c} c': o_i \in pt(c^t{:}m_{this}) \\ c'': o_u \in pt(c^t{:}m_{pj}), 1 \le j \le n \\ c''': o_v \in pt(c{:}r) \end{array}}$$

Caller context: c

Rule: Call

Kind	Statement	Rule (under context c)
Call	`l: r = x.k(a1,…,an)`	$\dfrac{c': o_i \in pt(c: x),\ m = \text{Dispatch}(o_i, \mathrm{k}),\ c^t = \text{Select}(c, l, c': o_i),\ c'': o_u \in pt(c: aj), 1 \le j \le n,\ c''': o_v \in pt(c^t: m_{ret})}{c': o_i \in pt(c^t: m_{this}),\ c'': o_u \in pt\left(c^t: m_{pj}\right), 1 \le j \le n,\ c''': o_v \in pt(c: r)}$

- $\text{Dispatch}(o_i, \mathrm{k})$: resolves the virtual dispatch of k on o_i to a target method (based on type of o_i)

Rule: Call

Caller context: c
Callee context: c^t

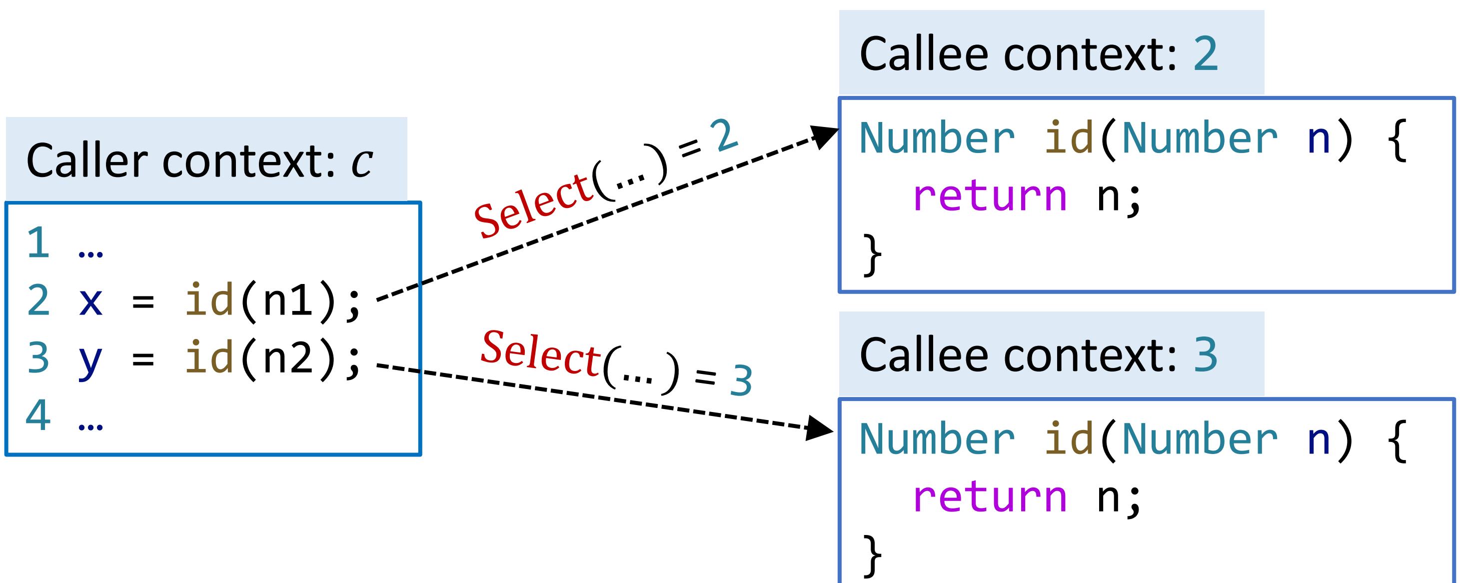
Kind	Statement	Rule (under context c)
Call	`l: r = x.k(a1,…,an)`	$\dfrac{\begin{array}{c} c': o_i \in pt(c{:}x), \\ m = \text{Dispatch}(o_i, \text{k}), \boxed{c^t = \text{Select}(c, l, c': o_i)} \\ c'': o_u \in pt(c{:}aj), 1 \le j \le n \\ c''': o_v \in pt(c^t: m_{ret}) \end{array}}{\begin{array}{c} c': o_i \in pt(c^t: m_{this}) \\ c'': o_u \in pt(c^t: m_{pj}), 1 \le j \le n \\ c''': o_v \in pt(c{:}r) \end{array}}$

- $\text{Dispatch}(o_i, \text{k})$: resolves the virtual dispatch of k on o_i to a target method (based on type of o_i)
- $\text{Select}(c, l, c': o_i)$: selects context for target method m, based on the information available at call site l

Rule: Call

Caller context: c
Callee context: c^t

Kind	Statement	Rule (under context c)
Call	`l: r = x.k(a1,…,an)`	$\dfrac{\begin{array}{c} c':o_i \in pt(c:x), \\ m = \text{Dispatch}(o_i, \text{k}), \boxed{c^t = \text{Select}(c, l, c':o_i)} \\ c'':o_u \in pt(c:aj), 1 \le j \le n \\ c''':o_v \in pt(c^t:m_{ret}) \end{array}}{\begin{array}{c} c':o_i \in pt(c^t:m_{this}) \\ c'':o_u \in pt\left(c^t:m_{pj}\right), 1 \le j \le n \\ c''':o_v \in pt(c:r) \end{array}}$



Rule: Call

Caller context: c
Callee context: c^t

Kind	Statement	Rule (under context c)
Call	`l: r = x.k(a1,…,an)`	$\dfrac{c': o_i \in pt(c:x), \quad m = \text{Dispatch}(o_i, \text{k}), c^t = \text{Select}(c, l, c':o_i) \quad c'': o_u \in pt(c:aj), 1 \le j \le n \quad c''': o_v \in pt(c^t: m_{ret})}{c': o_i \in pt(c^t: m_{this}) \quad c'': o_u \in pt(c^t: m_{pj}), 1 \le j \le n \quad c''': o_v \in pt(c:r)}$

- $\text{Dispatch}(o_i, \text{k})$: resolves the virtual dispatch of k on o_i to a target method (based on type of o_i)
- $\text{Select}(c, l, c':o_i)$: selects context for target method m, based on the information available at call site l

➢ $c^t: m_{this}$: this variable of $c^t: m$

Rule: Call

Caller context: c
Callee context: c^t

Kind	Statement	Rule (under context c)
Call	`l: r = x.k(a1,…,an)`	$$\frac{\begin{array}{c} c':o_i \in pt(c:x), \\ m = \text{Dispatch}(o_i, \mathrm{k}), c^t = \text{Select}(c, l, c':o_i) \\ c'':o_u \in pt(c:aj), 1 \leq j \leq n \\ c''':o_v \in pt(c^t:m_{ret}) \end{array}}{\begin{array}{c} c':o_i \in pt(c^t:m_{this}) \\ c'':o_u \in pt\left(c^t:m_{pj}\right), 1 \leq j \leq n \\ c''':o_v \in pt(c:r) \end{array}}$$

- $\text{Dispatch}(o_i, \mathrm{k})$: resolves the virtual dispatch of k on o_i to a target method (based on type of o_i)
- $\text{Select}(c, l, c':o_i)$: selects context for target method m, based on the information available at call site l
- $c^t:m_{this}$: this variable of $c^t:m$
- ➢ $c^t:m_{pj}$: the *j*-th parameter of $c^t:m$

Rule: Call

Caller context: c
Callee context: c^t

Kind	Statement	Rule (under context c)
Call	`l: r = x.k(a1,…,an)`	$\dfrac{\begin{array}{c} c':o_i \in pt(c{:}x), \\ m = \text{Dispatch}(o_i, \text{k}), c^t = \text{Select}(c, l, c':o_i) \\ c'':o_u \in pt(c{:}aj), 1 \le j \le n \\ c''':o_v \in pt(c^t{:}m_{ret}) \end{array}}{\begin{array}{c} c':o_i \in pt(c^t{:}m_{this}) \\ c'':o_u \in pt(c^t{:}m_{pj}), 1 \le j \le n \\ c''':o_v \in pt(c{:}r) \end{array}}$

Caller context: c

```
1 …
2 x = id(n1);
3 y = id(n2);
4 …
```

Select(…) = 2

Select(…) = 3

Callee context: 2

```
Number id(Number n) {
  return n;
}
```

Callee context: 3

```
Number id(Number n) {
  return n;
}
```

Rule: Call

Caller context: c
Callee context: c^t

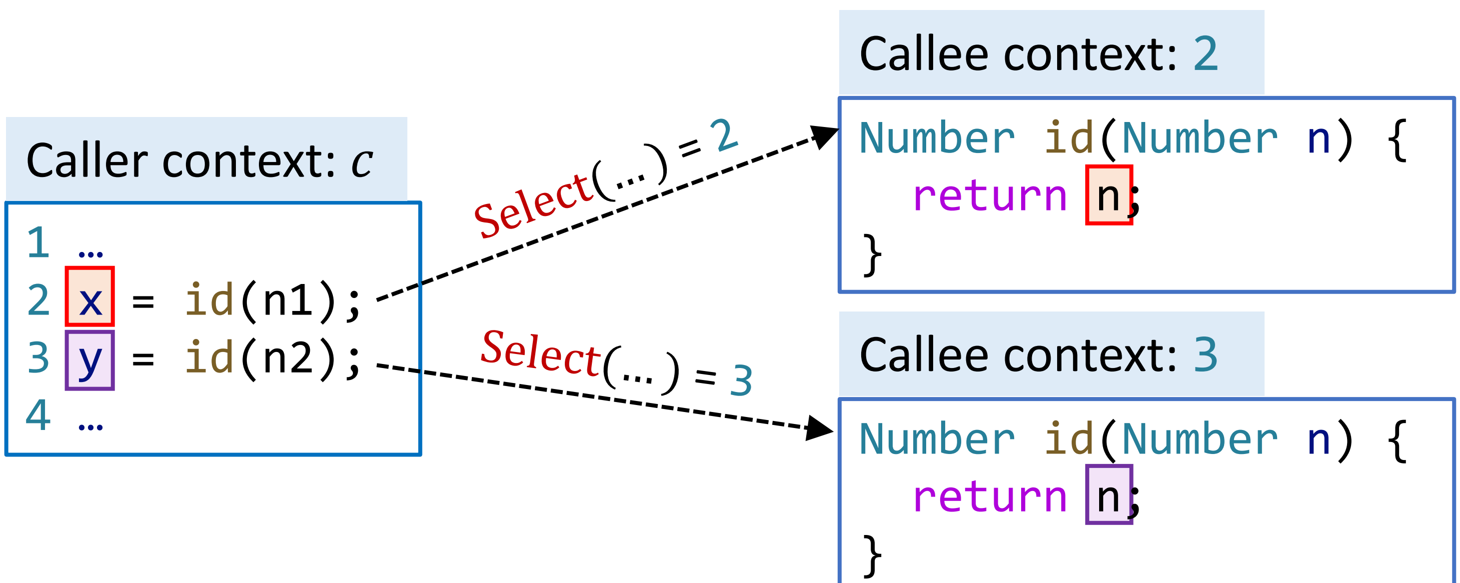
Kind	Statement	Rule (under context c)
Call	`l: r = x.k(a1,…,an)`	$\dfrac{c':o_i \in pt(c:x),\quad m = \text{Dispatch}(o_i, \text{k}), c^t = \text{Select}(c, l, c':o_i),\quad c'':o_u \in pt(c:aj), 1 \le j \le n,\quad c''':o_v \in pt(c^t:m_{ret})}{c':o_i \in pt(c^t:m_{this}),\quad c'':o_u \in pt(c^t:m_{pj}), 1 \le j \le n,\quad c''':o_v \in pt(c:r)}$

- $\text{Dispatch}(o_i, \text{k})$: resolves the virtual dispatch of k on o_i to a target method (based on type of o_i)
- $\text{Select}(c, l, c':o_i)$: selects context for target method m, based on the information available at call site l
- $c^t:m_{this}$: this variable of $c^t:m$
- $c^t:m_{pj}$: the *j*-th parameter of $c^t:m$
- ➢ $c^t:m_{ret}$: the variable that holds the return value of $c^t:m$

Rule: Call

Caller context: c
Callee context: c^t

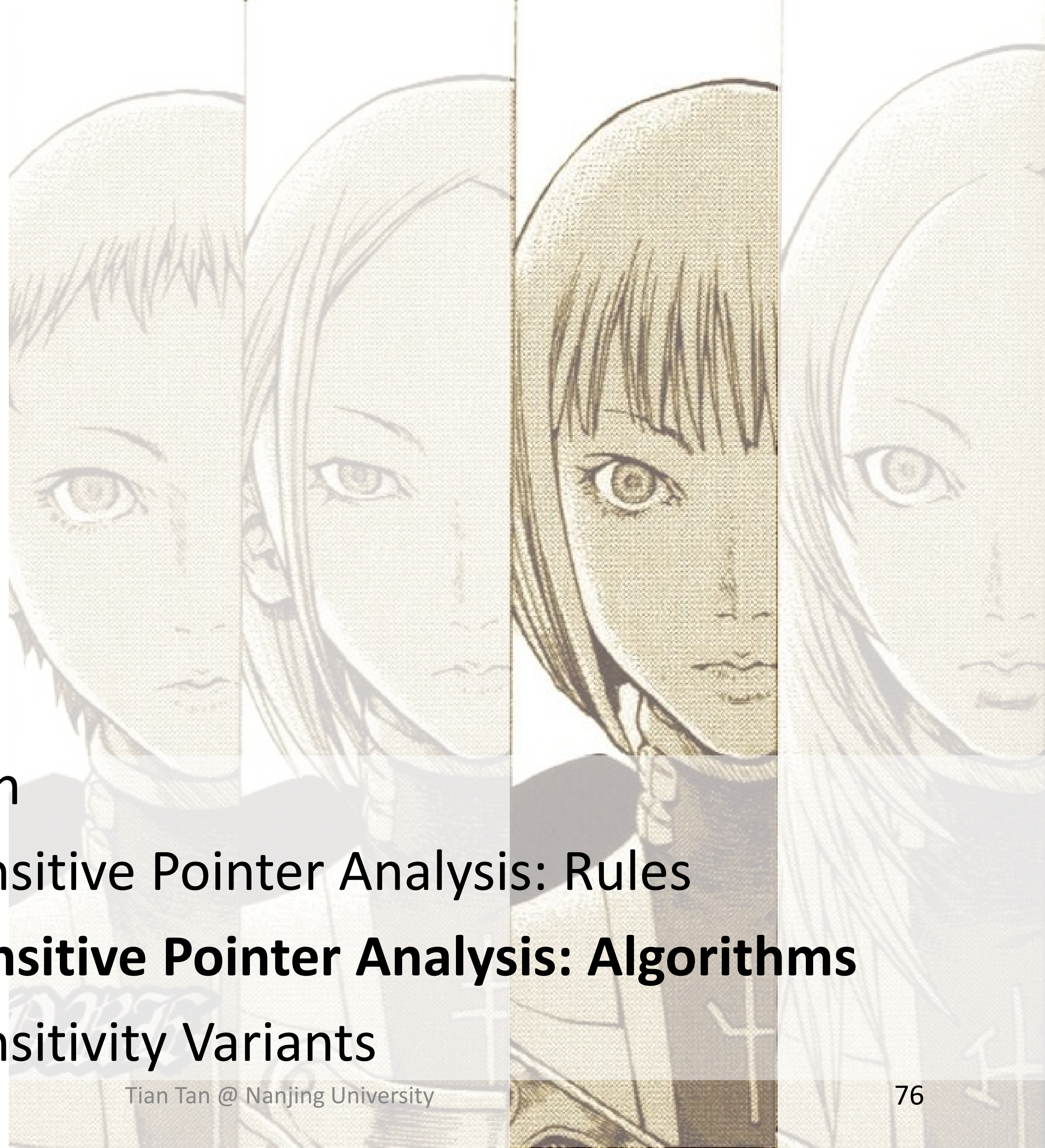
Kind	Statement	Rule (under context c)
Call	`l: r = x.k(a1,…,an)`	$\dfrac{\begin{array}{c} c':o_i \in pt(c:x), \\ m = \text{Dispatch}(o_i, \text{k}), c^t = \text{Select}(c, l, c':o_i) \\ c'':o_u \in pt(c:aj), 1 \le j \le n \\ c''':o_v \in pt(c^t:m_{ret}) \end{array}}{\begin{array}{c} c':o_i \in pt(c^t:m_{this}) \\ c'':o_u \in pt(c^t:m_{pj}), 1 \le j \le n \\ c''':o_v \in pt(c:r) \end{array}}$



The X You Need To Understand in This Lecture

- Concept of context sensitivity (C.S.)
- Concept of context-sensitive heap (C.S. heap)
- Why C.S. and C.S. heap improve precision
- Context-sensitive pointer analysis rules

Contents



How to Implement Context-Sensitive Pointer Analysis

- Recall context-insensitive pointer analysis

1. Build pointer flow graph (PFG)

Mutually dependent

2. Propagate points-to information on PFG

How to Implement Context-Sensitive Pointer Analysis

- Recall context-insensitive pointer analysis

1. Build pointer flow graph (PFG)

Mutually dependent

2. Propagate points-to information on PFG

- Context-sensitive pointer analysis

1. Build pointer flow graph (PFG with C.S.)

Mutually dependent

2. Propagate points-to information on PFG with C.S.

Pointer Flow Graph with C.S.

Pointer flow graph of a program is a *directed graph* that expresses how objects flow among the pointers in the program

Pointer Flow Graph with C.S.

Pointer flow graph of a program is a *directed graph* that expresses how objects flow among the pointers in the program

- Nodes: $\text{CSPointer} = (C \times V) \cup (C \times O \times F)$

  A node n represents *a context-sensitive variable* or *a field of a context-sensitive abstract object*

  With C.S., the nodes (pointers) are qualified by contexts

Pointer Flow Graph with C.S.

Pointer flow graph of a program is a *directed graph* that expresses how objects flow among the pointers in the program

- Nodes: CSPointer = (C × V) ∪ (C × O × F)

  A node n represents *a context-sensitive variable* or *a field of a context-sensitive abstract object*

  With C.S., the nodes (pointers) are qualified by contexts

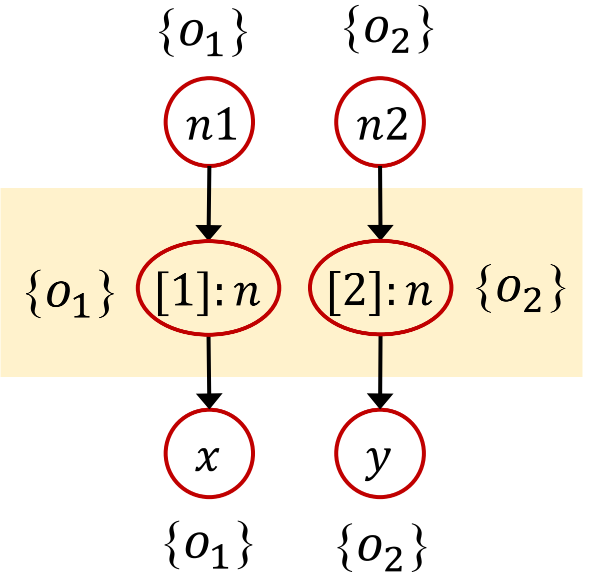
- Edges: CSPointer × CSPointer

  An edge $x \rightarrow y$ means that the objects pointed by pointer x *may flow to* (and also be pointed to by) pointer y

e.g., edge $c{:}\,a \rightarrow c'{:}\,b$ means that objects in $pt(c{:}\,a)$ may flow to $pt(c'{:}\,b)$

An Example

```
x = id(n1);
y = id(n2);
int i = x.get();

Number id(Number n) {
  return n;
}
```



The PFG contains two nodes for variable n in method `id()`, one node per context

Pointer Flow Graph: Edges

- PFG edges are added according to the statements of the program and the corresponding rules

Kind	Statement	Rule (under context c)	PFG Edge
New	`i: x = new T()`	$\frac{}{c: o_i \in pt(c: x)}$	N/A
Assign	`x = y`	$\frac{c': o_i \in pt(\mathbf{c: y})}{c': o_i \in pt(\mathbf{c: x})}$	$c: x \leftarrow c: y$
Store	`x.f = y`	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(\mathbf{c: y})}{c'': o_j \in pt(\mathbf{c': o_i.f})}$	$c': o_i.f \leftarrow c: y$
Load	`y = x.f`	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(\mathbf{c': o_i.f})}{c'': o_j \in pt(\mathbf{c: y})}$	$c: y \leftarrow c': o_i.f$

Pointer Flow Graph: Call

Kind	Statement	Rule (under context c)
Call	`l: r = x.k(a1,…,an)`	$\dfrac{\begin{array}{c} c':o_i \in pt(c:x), \\ m = \text{Dispatch}(o_i, \text{k}), c^t = \text{Select}(c, l, c':o_i) \\ c'':o_u \in pt(\boldsymbol{c}: \boldsymbol{aj}), 1 \le j \le n \\ c''':o_v \in pt(\boldsymbol{c^t}: \boldsymbol{m_{ret}}) \end{array}}{\begin{array}{c} c':o_i \in pt(c^t:m_{this}) \\ c'':o_u \in pt(\boldsymbol{c^t}: \boldsymbol{m_{pj}}), 1 \le j \le n \\ c''':o_v \in pt(\boldsymbol{c}: \boldsymbol{r}) \end{array}}$

Caller context: c

```
C x = new T();
…
r = x.foo(a1, a2);
```

Callee context: c^t

```
class T … {   …
  B foo(A p1, A p2) {
    this…
    return ret;
}}
```

PFG Edge
$c:a1 \rightarrow c^t:m^{p1}$
…
$c:an \rightarrow c^t:m_{pn}$
$c:r \leftarrow c^t:m_{ret}$

C.S. Pointer Analysis: Algorithm

Solve(m^{entry})
  WL=[ ], PFG={}, S={}, RM={}, CG={}
  AddReachable([ ]: m^{entry})
  while WL is not empty **do**
    remove $\langle n, pts \rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c{:}x$ **then**
      foreach $c'{:}o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c{:}y$, $c'{:}o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c'{:}o_i.f$, $c{:}y$)
        ProcessCall($c{:}x$, $c'{:}o_i$)

S	Set of reachable statements
S_m	Set of statements in method m
RM	Set of **C.S.** reachable methods
CG	**C.S.** call graph edges

AddReachable($c{:}m$)
  if $c{:}m \notin RM$ **then**
    add $c{:}m$ to RM
    $S \cup= S_m$
    foreach `i: x = new T()` $\in S_m$ **do**
      add $\langle c{:}x, \{c{:}o_i\} \rangle$ to WL
    foreach `x = y` $\in S_m$ **do**
      AddEdge($c{:}y$, $c{:}x$)

ProcessCall($c{:}x$, $c'{:}o_i$)
  foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
    m = Dispatch(o_i, k)
    c^t = Select($c, l, c'{:}o_i$)
    add $\langle c^t{:}m_{this}, \{c'{:}o_i\} \rangle$ to WL
    if $c{:}l \rightarrow c^t{:}m \notin CG$ **then**
      add $c{:}l \rightarrow c^t{:}m$ to CG
      AddReachable($c^t{:}m$)
      foreach parameter p_i of m **do**
        AddEdge($c{:}a_i$, $c^t{:}p_i$)
      AddEdge($c^t{:}m_{ret}$, $c{:}r$)

C.I. Pointer Analysis: Algorithm

Solve(m^{entry})
    WL=[ ], PFG={}, S={}, RM={}, CG={}
    AddReachable(m^{entry})
    while WL is not empty **do**
        remove $\langle n, pts \rangle$ from WL
        $\Delta = pts - pt(n)$
        Propagate(n, Δ)
        if n represents a variable x **then**
            foreach $o_i \in \Delta$ **do**
                foreach `x.f = y` $\in S$ **do**
                    AddEdge(y, $o_i.f$)
                foreach `y = x.f` $\in S$ **do**
                    AddEdge($o_i.f$, y)
                ProcessCall(x, o_i)

AddReachable(m)
    if $m \notin RM$ **then**
        add m to RM
        $S \cup= S_m$
        foreach `i: x = new T()` $\in S_m$ **do**
            add $\langle x, \{o_i\} \rangle$ to WL
        foreach `x = y` $\in S_m$ **do**
            AddEdge(y, x)

ProcessCall(x, o_i)
    foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
        m = Dispatch(o_i, k)
        add $\langle m_{this}, \{o_i\} \rangle$ to WL
        if $l \to m \notin CG$ **then**
            add $l \to m$ to CG
            AddReachable($: m$)
            foreach parameter p_i of m **do**
                AddEdge(a_i, p_i)
            AddEdge(m_{ret}, r)

S	Set of reachable statements
S_m	Set of statements in method m
RM	Set of **C.I.** reachable methods
CG	**C.I.** call graph edges

C.S. Pointer Analysis: Algorithm

Solve(m^{entry})
  WL=[ ], PFG={}, S={}, RM={}, CG={}
  AddReachable($[\,]: m^{entry}$)
  while WL is not empty **do**
    remove $\langle n, pts\rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c: x$ **then**
      foreach $c': o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c: y$, $c': o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c': o_i.f$, $c: y$)
        ProcessCall($c: x$, $c': o_i$)

S	Set of reachable statements
S_m	Set of statements in method m
RM	Set of **C.S.** reachable methods
CG	**C.S.** call graph edges

C.S. Pointer Analysis: Algorithm

Solve(m^{entry})
  WL=[ ],PFG={},S={},RM={},CG={}
  AddReachable($[\,]: m^{entry}$)
  while WL is not empty **do**
    remove $\langle n, pts \rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c: x$ **then**
      foreach $c': o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c: y$, $c': o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c': o_i.f$, $c: y$)
        ProcessCall($c: x$, $c': o_i$)

S	Set of reachable statements
S_m	Set of statements in method m
RM	Set of **C.S.** reachable methods
CG	**C.S.** call graph edges

Callee context: c^t

```
class T … {   …
  B foo(A p1, A p2) {
    this…
    return ret;
  }}
```

c^t: T.foo(A, A) $\in RM$

C.S. Pointer Analysis: Algorithm

Solve(m^{entry})
  WL=[ ], PFG={}, S={}, RM={}, CG={}
  AddReachable($[\,]: m^{entry}$)
  while WL is not empty **do**
    remove $\langle n, pts\rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c{:}\,x$ **then**
      foreach $c'{:}\,o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c{:}\,y$, $c'{:}\,o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c'{:}\,o_i.f$, $c{:}\,y$)
        ProcessCall($c{:}\,x$, $c'{:}\,o_i$)

S	Set of reachable statements
S_m	Set of statements in method m
RM	Set of **C.S.** reachable methods
CG	**C.S.** call graph edges

Caller context: c

```
1 C x = new T();
2 r = x.foo(a1, a2);
```

Callee context: c^t

```
class T … {
  B foo(A p1, A p2) {
    this…
    return ret;
  }}
```

$c^t{:}\,\mathrm{T.foo(A, A)} \in RM$

$c{:}\,2 \rightarrow c^t{:}\,\mathrm{T.foo(A, A)} \in CG$

C.S. Pointer Analysis: Algorithm

Solve(m^{entry})
    WL=[ ], PFG={}, S={}, RM={}, CG={}
    AddReachable([ ]: m^{entry})
    while WL is not empty **do**
        remove $\langle n, pts \rangle$ from WL
        $\Delta = pts - pt(n)$
        Propagate(n, Δ)
        if n represents a variable c: x **then**
            foreach c': $o_i \in \Delta$ **do**
                foreach `x.f = y` $\in S$ **do**
                    AddEdge(c: y, c': $o_i.f$)
                foreach `y = x.f` $\in S$ **do**
                    AddEdge(c': $o_i.f$, c: y)
                ProcessCall(c: x, c': o_i)

AddReachable(c: m)
    if c: $m \notin RM$ **then**
        add c: m to RM
        $S \cup= S_m$
        foreach `i: x = new T()` $\in S_m$ **do**
            add $\langle c{:}\,x, \{c{:}\,o_i\}\rangle$ to WL
        foreach `x = y` $\in S_m$ **do**
            AddEdge(c: y, c: x)

Kind	Statement	Rule (under context c)	PFG Edge
New	`i: x = new T()`	$\frac{}{c{:}\,o_i \in pt(c{:}\,x)}$	N/A
Assign	`x = y`	$\frac{c'{:}\,o_i \in pt(c{:}\,y)}{c'{:}\,o_i \in pt(c{:}\,x)}$	$c{:}\,x \leftarrow c{:}\,y$

C.S. Pointer Analysis: Algorithm

Solve(m^{entry})
  WL=[ ],PFG={},S={},RM={},CG={}
  AddReachable([ ]: m^{entry})
  while WL is not empty **do**
    remove $\langle n, pts \rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable c: x **then**
      foreach c': $o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge(c: y, c': $o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge(c': $o_i.f$, c: y)
        ProcessCall(c: x, c': o_i)

AddEdge(s, t)
  if $s \rightarrow t \notin PFG$ **then**
    add $s \rightarrow t$ to PFG
    if $pt(s)$ is not empty **then**
      add $\langle t, pt(s) \rangle$ to WL

Propagate(n, pts)
  if pts is not empty **then**
    $pt(n)$ ⋃= pts
    foreach $n \rightarrow s \in PFG$ **do**
      add $\langle s, pts \rangle$ to WL

Exactly same as in C.I. analysis

S	Set of reachable statements
S_m	Set of statements in method m
RM	Set of **C.S.** reachable methods
CG	**C.S.** call graph edges

C.S. Pointer Analysis: Algorithm

Solve(m^{entry})
  WL=[ ],PFG={},S={},RM={},CG={}
  AddReachable([ ]: m^{entry})
  while WL is not empty **do**
    remove $\langle n, pts \rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c{:}\,x$ **then**
      foreach $c'{:}\,o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c{:}\,y$, $c'{:}\,o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c'{:}\,o_i.f$, $c{:}\,y$)
        ProcessCall($c{:}\,x$, $c'{:}\,o_i$)

AddEdge(s, t)
  if $s \rightarrow t \notin PFG$ **then**
    add $s \rightarrow t$ to PFG
    if $pt(s)$ is not empty **then**
      add $\langle t, pt(s) \rangle$ to WL

Propagate(n, pts)
  if pts is not empty **then**
    $pt(n) \cup= pts$
    foreach $n \rightarrow s \in PFG$ **do**
      add $\langle s, pts \rangle$ to WL

Kind	Statement	Rule (under context c)	PFG Edge
Store	`x.f = y`	$\dfrac{c'{:}\,o_i \in pt(c{:}\,x),\ c''{:}\,o_j \in pt(\boldsymbol{c{:}\,y})}{c''{:}\,o_j \in pt(\boldsymbol{c'{:}\,o_i.f})}$	$c'{:}\,o_i.f \leftarrow c{:}\,y$
Load	`y = x.f`	$\dfrac{c'{:}\,o_i \in pt(c{:}\,x),\ c''{:}\,o_j \in pt(\boldsymbol{c'{:}\,o_i.f})}{c''{:}\,o_j \in pt(\boldsymbol{c{:}\,y})}$	$c{:}\,y \leftarrow c'{:}\,o_i.f$

C.S. Pointer Analysis: Algorithm

Solve(m^{entry})
  WL=[ ],PFG={},S={},RM={},CG={}
  AddReachable([ ]: m^{entry})
  while WL is not empty **do**
    remove $\langle n, pts\rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c\!: x$ **then**
      foreach $c'\!: o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c\!: y$, $c'\!: o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c'\!: o_i.f$, $c\!: y$)
        ProcessCall($c\!: x$, $c'\!: o_i$)

ProcessCall($c\!: x$, $c'\!: o_i$)
  foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
    m = Dispatch(o_i, k)
    c^t = Select($c, l, c'\!: o_i$)
    add $\langle c^t\!: m_{this}, \{c'\!: o_i\}\rangle$ to WL
    if $c\!: l \to c^t\!: m \notin CG$ **then**
      add $c\!: l \to c^t\!: m$ to CG
      AddReachable($c^t\!: m$)
      foreach parameter p_i of m **do**
        AddEdge($c\!: a_i$, $c^t\!: p_i$)
      AddEdge($c^t\!: m_{ret}$, $c\!: r$)

Kind	Statement	Rule (under context c)	PFG Edge
Call	`l: r = x.k(a1,…,an)`	$\dfrac{\begin{array}{c} c'\!: o_i \in pt(c\!: x), \\ m = \text{Dispatch}(o_i, \text{k}), c^t = \text{Select}(c, l, c'\!: o_i) \\ c''\!: o_u \in pt(\boldsymbol{c\!: aj}), 1 \le j \le n \\ c'''\!: o_v \in pt(\boldsymbol{c^t\!: m_{ret}}) \end{array}}{\begin{array}{c} c'\!: o_i \in pt(c^t\!: m_{this}) \\ c''\!: o_u \in pt(\boldsymbol{c^t\!: m_{pj}}), 1 \le j \le n \\ c'''\!: o_v \in pt(\boldsymbol{c\!: r}) \end{array}}$	$c\!: a1 \to c^t\!: m^{p1}$ … $c\!: an \to c^t\!: m_{pn}$ $c\!: r \leftarrow c^t\!: m_{ret}$

C.S. Pointer Analysis: Algorithm

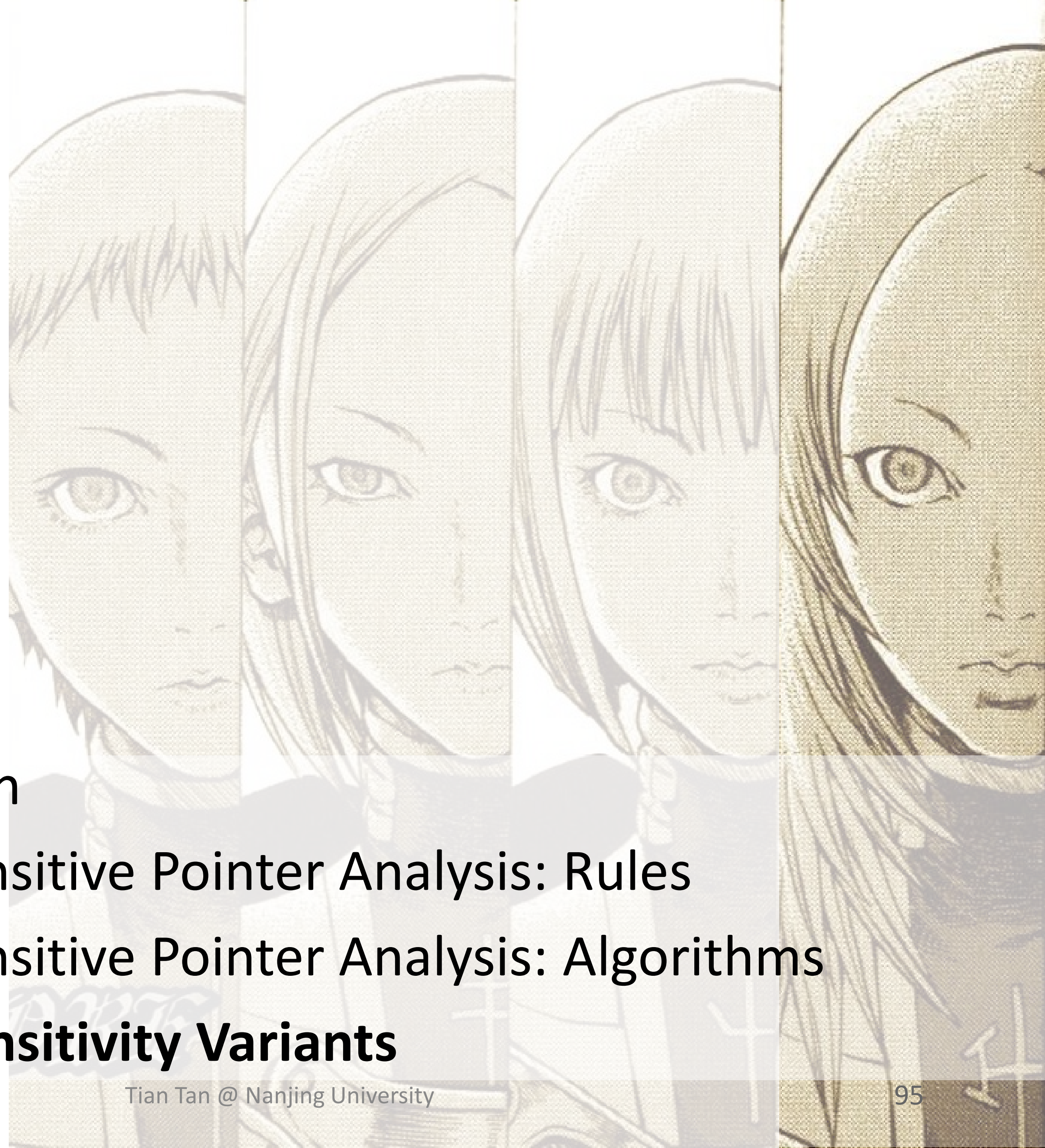
Solve(m^{entry})
    WL=[ ],PFG={},S={},RM={},CG={}
    AddReachable([ ]: m^{entry})
    while WL is not empty **do**
        remove $\langle n, pts\rangle$ from WL
        $\Delta = pts - pt(n)$
        Propagate(n, Δ)
        if n represents a variable c: x **then**
            foreach c': $o_i \in \Delta$ **do**
                foreach `x.f = y` $\in S$ **do**
                    AddEdge(c: y, c': $o_i.f$)
                foreach `y = x.f` $\in S$ **do**
                    AddEdge(c': $o_i.f$, c: y)
                ProcessCall(c: x, c': o_i)

S	Set of reachable statements
S_m	Set of statements in method m
RM	Set of **C.S.** reachable methods
CG	**C.S.** call graph edges

AddReachable(c: m)
    if c: $m \notin RM$ **then**
        add c: m to RM
        $S \cup= S_m$
        foreach `i: x = new T()` $\in S_m$ **do**
            add $\langle c$: x, {c: o_i}$\rangle$ to WL
        foreach `x = y` $\in S_m$ **do**
            AddEdge(c: y, c: x)

ProcessCall(c: x, c': o_i)
    foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
        m = Dispatch(o_i, k)
        c^t = Select(c, l, c': o_i)
        add $\langle c^t$: m_{this}, {c': o_i}$\rangle$ to WL
        if c: $l \rightarrow c^t$: $m \notin CG$ **then**
            add c: $l \rightarrow c^t$: m to CG
            AddReachable(c^t: m)
            foreach parameter p_i of m **do**
                AddEdge(c: a_i, c^t: p_i)
            AddEdge(c^t: m_{ret}, c: r)

Contents



Context Sensitivity Variants

- Call-site sensitivity
- Object sensitivity
- Type sensitivity
- ……

Context Sensitivity Variants

- Call-site sensitivity
- Object sensitivity
- Type sensitivity
- ……

$$\text{Select}(\underbrace{c}_{\text{caller context}},\ \underbrace{l}_{\text{call site}},\ \underbrace{c':o_i}_{\text{receiver object, with heap context}})$$

Context Sensitivity Variants

- Call-site sensitivity
- Object sensitivity
- Type sensitivity
- ……

$\textcolor{red}{\text{Select}}(c, l, c' : o_i)$

caller context
c

call site
2

receiver object, with heap context
$c : o_1$

Caller context: c

```
1 C x = new T();
2 r = x.foo(a1, a2);
```

Callee context: c^t

```
class T … {
  B foo(A p1, A p2) {
    this…
    return ret;
  }}
```

Context Insensitivity

- Can be seen as a special case of context sensitivity in C.S. analysis framework

$$\color{red}{\mathrm{Select}}(\_,\_,\_) = [\,]$$

Call-Site Sensitivity*

- Each context consists of a list of call sites (call chain)
  - At a method call, append the call site to the caller context as callee context
  - Essentially the abstraction of call stacks

$$\text{Select}(\boldsymbol{c}, \boldsymbol{l}, \_) = [l', \ldots, l'', l]$$
$$\text{where } c = [l', \ldots, l'']$$

Also called call-string sensitivity, or *k*-CFA

* Olin Shivers, 1991. "*Control-Flow Analysis of Higher-Order Languages*". Ph.D. Dissertation. Carnegie Mellon University.

Call-Site Sensitivity: Example

```
void main() {
  …
  a.foo();
  …
}

void foo() {
  …
  b.bar();
  …
}

void bar() {
  …
  …
  …
}
```

Call-Site Sensitivity: Example

```
void main() {
  …
  a.foo();
  …
}

void foo() {
  …
  b.bar();
  …
}

void bar() {
  …
  …
  …
}
```

Context: [ ]

Call-Site Sensitivity: Example

```
1  void main() {
2    …
3    a.foo();
4    …
5  }
6
7  void foo() {
8    …
9    b.bar();
10   …
11 }
12
13 void bar() {
14   …
15   …
16   …
17 }
```

Context: [ ]

Context: [3]

Call-Site Sensitivity: Example

Context: [ ]

```
1  void main() {
2    …
3    a.foo();
4    …
5  }
6
```

Context: [3]

```
7  void foo() {
8    …
9    b.bar();
10   …
11 }
12
```

Context: [3,9]

```
13 void bar() {
14   …
15   …
16   …
17 }
```

Call-Site Sensitivity: Example

```
1  void main() {
2    …
3    a.foo();
4    …
5  }
6
7  void foo() {
8    …
9    b.bar();
10   …
11 }
12
13 void bar() {
14   …
15   bar(); ?
16   …
17 }
```

Context: [ ]

Context: [3]

Context: [3,9]

Call-Site Sensitivity: Example

```
1   void main() {
2     …
3     a.foo();
4     …
5   }
6
7   void foo() {
8     …
9     b.bar();
10    …
11  }
12
13  void bar() {
14    …
15    bar();
16    …
17  }
```

Context: [ ]

Context: [3]

Context: [3,9]
[3,9,15]
[3,9,15,15]
[3,9,15,15,15]
[3,9,15,15,15, …]
…

k-Limiting Context Abstraction

- Motivation
  - Ensure termination of pointer analysis
  - Avoid too many contexts (long call chains) in real-world programs blow up the pointer analysis

k-Limiting Context Abstraction

- Motivation
  - Ensure termination of pointer analysis
  - Avoid too many contexts (long call chains) in real-world programs blow up the pointer analysis

- Approach: set an upper bound for length of contexts, denoted by *k*
  - For call-site sensitivity, each context consists of the last *k* call sites of the call chains
  - In practice, *k* is a small number (usually ≤3)
  - Method contexts and heap contexts may use different *k*
    - e.g., *k*=2 for method context, *k*=1 for heap contexts

k-Call-Site Sensitivity/*k*-CFA

- 1-call-site/1-CFA

$$\text{Select}(\_, \boldsymbol{l}, \_) = [l]$$

k-Call-Site Sensitivity/*k*-CFA

- 1-call-site/1-CFA

$$\textcolor{red}{\text{Select}}(\_, \boldsymbol{l}, \_) = [l]$$

```
1  void main() {
2     …
3     a.foo();
4     …
5  }
6
7  void foo() {
8     …
9     b.bar();
10    …
11 }
12
13 void bar() {
14    …
15    bar();
16    …
17 }
```

Context: [ ]

Context: [3]

Context: ? ? ?

k-Call-Site Sensitivity/k-CFA

- 1-call-site/1-CFA

$$\textcolor{red}{\text{Select}}(\_, \boldsymbol{l}, \_) = [l]$$

```
1  void main() {
2    …
3    a.foo();
4    …
5  }
6
7  void foo() {
8    …
9    b.bar();
10   …
11 }
12
13 void bar() {
14   …
15   bar();
16   …
17 }
```

Context: [ ]

Context: [3]

Context: [9]
[15]

k-Call-Site Sensitivity/k-CFA

- 1-call-site/1-CFA

$$\text{Select}(\_, \boldsymbol{l}, \_) = [l]$$

- 2-call-site/2-CFA

$$\text{Select}(\boldsymbol{c}, \boldsymbol{l}, \_) = [l'', l]$$
$$\text{where } c = [l', l'']$$

1-Call-Site: Example

Solve(m^{entry})
  WL=[ ],PFG={},S={},RM={},CG={}
  AddReachable([ ]: m^{entry})
  while WL is not empty **do**
    remove $\langle n, pts \rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c{:}\,x$ **then**
      foreach $c'{:}\,o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c{:}\,y$, $c'{:}\,o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c'{:}\,o_i.f$, $c{:}\,y$)
        ProcessCall($c{:}\,x$, $c'{:}\,o_i$)

```
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

1-Call-Site: Example

Solve(m^{entry})
  WL=[ ],PFG={},S={},RM={},CG={}
  AddReachable([ ]: m^{entry})
  while WL is not empty **do**
    remove $\langle n, pts\rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c{:}\,x$ **then**
      foreach $c'{:}\,o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c{:}\,y$, $c'{:}\,o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c'{:}\,o_i.f$, $c{:}\,y$)
        ProcessCall($c{:}\,x$, $c'{:}\,o_i$)

```java
interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

```java
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

1-Call-Site: Example

Solve(m^{entry})
  WL=[ ], PFG={}, S={}, RM={}, CG={}
  AddReachable([ ]: m^{entry})
  while WL is not empty **do**
    remove $\langle n, pts \rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c{:}\,x$ **then**
      foreach $c'{:}\,o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c{:}\,y$, $c'{:}\,o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c'{:}\,o_i.f$, $c{:}\,y$)
        ProcessCall($c{:}\,x$, $c'{:}\,o_i$)

```
interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

```
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get(); ?
  }
}
```

1-Call-Site: Example

Solve(m^{entry})
  WL=[ ], PFG={}, S={}, RM={}, CG={}
  AddReachable([ ]: m^{entry})
  while WL is not empty **do**
    remove $\langle n, pts \rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c{:}\,x$ **then**
      foreach $c'{:}\,o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c{:}\,y$, $c'{:}\,o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c'{:}\,o_i.f$, $c{:}\,y$)
        ProcessCall($c{:}\,x$, $c'{:}\,o_i$)

```
interface Number {
  int get(); }
class One implements Number {
  public int get() { return 1; }}
class Two implements Number {
  public int get() { return 2; }}
```

```
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

For simplicity, we do not apply C.S. heap and omit `this` variable of `C.id(Number)`

1-Call-Site: Example

WL: [ ]

Solve(m^{entry})
  WL=[ ], PFG={}, S={}, RM={}, CG={}
  AddReachable([ ]: m^{entry})
  while WL is not empty **do**
    remove $\langle n, pts\rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c{:}\,x$ **then**
      foreach $c'{:}\,o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c{:}\,y$, $c'{:}\,o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c'{:}\,o_i.f$, $c{:}\,y$)
        ProcessCall($c{:}\,x$, $c'{:}\,o_i$)

Processing:

PFG:

CG: { }

```java
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

RM: { }

1-Call-Site: Example

WL: [ ]

Solve(m^{entry})
  WL=[ ], PFG={}, S={}, RM={}, CG={}
  AddReachable([ ]: m^{entry})

Processing:

AddReachable(c: m)
  if c: $m \notin RM$ **then**
    add c: m to RM
    $S \cup= S_m$
    foreach `i: x = new T()` $\in S_m$ **do**
      add $\langle c{:}\,x,\ \{c{:}\,o_i\}\rangle$ to WL
    foreach `x = y` $\in S_m$ **do**
      AddEdge(c: y, c: x)

PFG:

CG: { }

```
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

RM: { }

1-Call-Site: Example

WL: [ ]

Solve(m^{entry})
  WL=[ ],PFG={},S={},RM={},CG={}
  AddReachable([ ]: m^{entry})

AddReachable(c: m)
  if c: $m \notin RM$ **then**
    add c: m to RM
    $S \cup= S_m$
    foreach `i: x = new T()` $\in S_m$ **do**
      add $\langle c: x, \{c: o_i\}\rangle$ to WL
    foreach `x = y` $\in S_m$ **do**
      AddEdge(c: y, c: x)

Processing:

PFG:

CG: { }

RM: { [ ]:C.main() }

```
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

1-Call-Site: Example

WL: $[\langle [\,]{:}c, \{o_3\}\rangle]$

Solve(m^{entry})
  WL=[ ], PFG={}, S={}, RM={}, CG={}
  AddReachable($[\,]: m^{entry}$)

Processing:

AddReachable($c: m$)
  if $c: m \notin RM$ **then**
    add $c: m$ to RM
    $S \cup= S_m$
    foreach `i: x = new T()` $\in S_m$ **do**
      add $\langle c{:}x, \{c{:}o_i\}\rangle$ to WL
    foreach `x = y` $\in S_m$ **do**
      AddEdge($c{:}y, c{:}x$)

PFG:

CG: { }

```
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

RM: { [ ]:C.main() }

1-Call-Site: Example

WL: [ ]

Processing: $\langle$[ ]$:c, \{o_3\}\rangle$

Solve(m^{entry})
  WL=[ ], PFG={}, S={}, RM={}, CG={}
  AddReachable([ ]$: m^{entry}$)
  while WL is not empty **do**
    remove $\langle n, pts\rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c{:}\,x$ **then**
      foreach $c'{:}\,o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c{:}\,y$, $c'{:}\,o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c'{:}\,o_i.f$, $c{:}\,y$)
        ProcessCall($c{:}\,x$, $c'{:}\,o_i$)

PFG:

CG: { }

RM: { [ ]:C.main() }

```
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

1-Call-Site: Example

WL: [ ]

Solve(m^{entry})
  WL=[ ], PFG={}, S={}, RM={}, CG={}
  AddReachable([ ]: m^{entry})
  while WL is not empty **do**
    remove $\langle n, pts\rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c: x$ **then**
      foreach $c': o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c: y$, $c': o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c': o_i.f$, $c: y$)
        ProcessCall($c: x$, $c': o_i$)

Processing: $\langle$[ ]:c, $\{o_3\}\rangle$

$\{o_3\}$

PFG: [ ]:c

CG: { }

```
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

RM: { [ ]:C.main() }

1-Call-Site: Example

WL: [ ]

$\text{ProcessCall}(c\!:x, c'\!:o_i)$
  foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
    $m = \text{Dispatch}(o_i, \text{k})$
    $c^t = \text{Select}(c, l, c'\!:o_i)$
    add $\langle c^t\!:m_{this}, \{c'\!:o_i\}\rangle$ to *WL*
    if $c\!:l \rightarrow c^t\!:m \notin CG$ **then**
      add $c\!:l \rightarrow c^t\!:m$ to CG
      $\text{AddReachable}(c^t\!:m)$
      foreach parameter p_i of m **do**
        $\text{AddEdge}(c\!:a_i,\ c^t\!:p_i)$
      $\text{AddEdge}(c^t\!:m_{ret}, c\!:r)$

$\text{ProcessCall}(c\!:x, \text{c}'\!:o_i)$

Processing: $\langle$[ ]:c, $\{o_3\}\rangle$

PFG: [ ]:c $\{o_3\}$

CG: { }

```
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

RM: { [ ]:C.main() }

1-Call-Site: Example

WL: [ ]

$\text{ProcessCall}(c{:}\,x, c'{:}\,o_i)$

foreach `l: r = x.k(a1,…,an)` $\in S$ **do**

$m = \text{Dispatch}(o_i, \mathsf{k})$

$c^t = \text{Select}(c, l, c'{:}\,o_i)$

add $\langle c^t{:}\,m_{this}, \{c'{:}\,o_i\}\rangle$ to WL

if $c{:}\,l \rightarrow c^t{:}\,m \notin CG$ **then**

add $c{:}\,l \rightarrow c^t{:}\,m$ to CG

$\text{AddReachable}(c^t{:}\,m)$

foreach parameter p_i of m **do**

$\text{AddEdge}(c{:}\,a_i,\ c^t{:}\,p_i)$

$\text{AddEdge}(c^t{:}\,m_{ret}, c{:}\,r)$

$m = \text{C.m()}$

$c^t =$?

Processing: $\langle[\,]{:}c, \{o_3\}\rangle$

PFG: $[\,]{:}c$ $\{o_3\}$

$\text{ProcessCall}(c{:}\,x, \mathrm{c}'{:}\,o_i)$

CG: { }

```
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

RM: { $[\,]$:C.main() }

1-Call-Site: Example

WL: [ ]

$\text{ProcessCall}(c\!:\!x, c'\!:\!o_i)$
  foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
    $m = \text{Dispatch}(o_i, \text{k})$
    $c^t = \text{Select}(c, l, c'\!:\!o_i)$
    add $\langle c^t\!:\!m_{this}, \{c'\!:\!o_i\}\rangle$ to *WL*
    if $c\!:\!l \to c^t\!:\!m \notin CG$ **then**
      add $c\!:\!l \to c^t\!:\!m$ to CG
      $\text{AddReachable}(c^t\!:\!m)$
      foreach parameter p_i of m **do**
        $\text{AddEdge}(c\!:\!a_i,\ c^t\!:\!p_i)$
      $\text{AddEdge}(c^t\!:\!m_{ret}, c\!:\!r)$

$m = \text{C.m()}$
$c^t = [4]$

$\text{ProcessCall}(c\!:\!x, \text{c}'\!:\!o_i)$

Processing: $\langle [\ ]\!:\!c, \{o_3\}\rangle$

PFG: $[\ ]\!:\!c$ $\{o_3\}$

CG: { }

RM: { [ ]:C.main() }

```
1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6
7   Number id(Number n) {
8     return n;
9   }
10  void m() {
11    Number n1,n2,x,y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }
```

1-Call-Site: Example

WL: $[\langle [4]\text{:C.}m_{this}, \{o_3\}\rangle]$

$\text{ProcessCall}(c\!:x, c'\!:o_i)$
- **foreach** `l: r = x.k(a1,…,an)` $\in S$ **do**
  - $m = \text{Dispatch}(o_i, \text{k})$
  - $c^t = \text{Select}(c, l, c'\!:o_i)$
  - add $\langle c^t\!:m_{this}, \{c'\!:o_i\}\rangle$ to *WL*
  - **if** $c\!:l \rightarrow c^t\!:m \notin CG$ **then**
    - add $c\!:l \rightarrow c^t\!:m$ to CG
    - $\text{AddReachable}(c^t\!:m)$
    - **foreach** parameter p_i of m **do**
      - $\text{AddEdge}(c\!:a_i,\ c^t\!:p_i)$
    - $\text{AddEdge}(c^t\!:m_{ret}, c\!:r)$

$m = \text{C.m()}$
$c^t = [4]$

Processing: $\langle [\ ]\!:c, \{o_3\}\rangle$

PFG: $[\ ]\!:c$ with $\{o_3\}$

$\text{ProcessCall}(c\!:x, \text{c}'\!:o_i)$

CG: { }

RM: { [ ]:C.main() }

```
1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6
7   Number id(Number n) {
8     return n;
9   }
10  void m() {
11    Number n1,n2,x,y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }
```

1-Call-Site: Example

WL: $[\langle [4]\text{:C.}m_{this}, \{o_3\}\rangle]$

ProcessCall(c: x, c': o_i)
  foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
    m = Dispatch(o_i, k)
    c^t = Select(c, l, c': o_i)
    add $\langle c^t: m_{this}, \{c': o_i\}\rangle$ to WL
    if $c: l \rightarrow c^t: m \notin CG$ **then**
      add $c: l \rightarrow c^t: m$ to CG
      AddReachable(c^t: m)
      foreach parameter p_i of m **do**
        AddEdge(c: a_i, c^t: p_i)
      AddEdge(c^t: m_{ret}, c: r)

m = C.m()
c^t = [4]

Processing: $\langle [\ ]: c, \{o_3\}\rangle$

PFG: $\{o_3\}$ $[\ ]: c$

ProcessCall(c: x, c′: o_i)

CG: $\{\ [\ ]\text{:4} \rightarrow [4]\text{:C.m()}\ \}$

```
1  class C {
2    static void main() {
3      C c = new C();
4      c.m();
5    }
6
7    Number id(Number n) {
8      return n;
9    }
10   void m() {
11     Number n1,n2,x,y;
12     n1 = new One();
13     n2 = new Two();
14     x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17   }
18 }
```

RM: $\{\ [\ ]\text{:C.main()}\ \}$

1-Call-Site: Example

WL: [⟨[4]:C.m_{this}, $\{o_3\}$⟩, ⟨[4]:$n1$, $\{o_{12}\}$⟩, ⟨[4]:$n2$, $\{o_{13}\}$⟩]

ProcessCall(c: x, c': o_i)
  foreach `l: r = x.k(a1,…,an)` ∈ S **do**
    m = Dispatch(o_i, k)
    c^t = Select(c, l, c': o_i)
    add ⟨c^t: m_{this}, $\{c'$: $o_i\}$⟩ to *WL*
    if c: $l \rightarrow c^t$: $m \notin CG$ **then**
      add c: $l \rightarrow c^t$: m to CG
      AddReachable(c^t: m)
      foreach parameter p_i of m **do**
        AddEdge(c: a_i, c^t: p_i)
      AddEdge(c^t: m_{ret}, c: r)

m = C.m()
c^t = [4]

ProcessCall(c: x, c′: o_i)

Processing: ⟨[ ]:c, $\{o_3\}$⟩

PFG:

AddReachable(c: m)
  if c: $m \notin RM$ **then**
    add c: m to RM
    S ∪= S_m
    foreach `i: x = new T()` ∈ S_m **do**
      add ⟨c: x, $\{c$: $o_i\}$⟩ to *WL*
    foreach `x = y` ∈ S_m **do**
      AddEdge(c: y, c: x)

```
class C {
  static void main() {
    C c = new C();
    c.m();
  }

  Number id(Number n) {
    return n;
  }
  void m() {
    Number n1,n2,x,y;
    n1 = new One();
    n2 = new Two();
    x = this.id(n1);
    y = this.id(n2);
    x.get();
  }
}
```

RM: { [ ]:C.main(), [4]:C.m() }

1-Call-Site: Example

WL: $[\langle [4]{:}n1, \{o_{12}\}\rangle, \langle [4]{:}n2, \{o_{13}\}\rangle]$

Solve(m^{entry})
  WL=[ ], PFG={}, S={}, RM={}, CG={}
  AddReachable([ ]: m^{entry})
  while WL is not empty **do**
    remove $\langle n, pts\rangle$ from WL
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable $c{:}\,x$ **then**
      foreach $c'{:}\,o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge($c{:}\,y$, $c'{:}\,o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge($c'{:}\,o_i.f$, $c{:}\,y$)
        ProcessCall($c{:}\,x$, $c'{:}\,o_i$)

Processing: $\langle [4]{:}C.m_{this}, \{o_3\}\rangle$

PFG:

$\{o_3\}$ $\boxed{[\ ]{:}c}$

$\{o_3\}$ $\boxed{[4]{:}C.m_{this}}$

CG: $\{ [\ ]{:}4 \rightarrow [4]{:}C.m() \}$

```
1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6
7   Number id(Number n) {
8     return n;
9   }
10  void m() {
11    Number n1,n2,x,y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }
```

RM: $\{ [\ ]{:}C.main(), [4]{:}C.m() \}$

1-Call-Site: Example

WL: $[\langle [4]{:}n1, \{o_{12}\}\rangle, \langle [4]{:}n2, \{o_{13}\}\rangle]$

ProcessCall($c{:}x, c'{:}o_i$)
  foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
    m = Dispatch(o_i, k)
    c^t = Select($c, l, c'{:}o_i$)
    add $\langle c^t{:}m_{this}, \{c'{:}o_i\}\rangle$ to *WL*
    if $c{:}l \rightarrow c^t{:}m \notin CG$ **then**
      add $c{:}l \rightarrow c^t{:}m$ to CG
      AddReachable($c^t{:}m$)
      foreach parameter p_i of m **do**
        AddEdge($c{:}a_i$, $c^t{:}p_i$)
      AddEdge($c^t{:}m_{ret}, c{:}r$)

→ ProcessCall($c{:}x$, $c'{:}o_i$)

Processing: $\langle [4]{:}\mathrm{C}.m_{this}, \{o_3\}\rangle$

PFG:

$\{o_3\}$
$[\ ]{:}c$

$\{o_3\}$
$[4]{:}\mathrm{C}.m_{this}$

CG: $\{\ [\ ]{:}4 \rightarrow [4]{:}\mathrm{C.m()}\ \}$

```
1  class C {
2    static void main() {
3      C c = new C();
4      c.m();
5    }
6
7    Number id(Number n) {
8      return n;
9    }
10   void m() {
11     Number n1,n2,x,y;
12     n1 = new One();
13     n2 = new Two();
14 →   x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17   }
18 }
```

RM: $\{\ [\ ]{:}\mathrm{C.main()}, [4]{:}\mathrm{C.m()}\ \}$

1-Call-Site: Example

WL: $[\langle [4]{:}n1, \{o_{12}\}\rangle, \langle [4]{:}n2, \{o_{13}\}\rangle]$

$\text{ProcessCall}(c{:}x, c'{:}o_i)$
  foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
    $m = \text{Dispatch}(o_i, \text{k})$
    $c^t = \text{Select}(c, l, c'{:}o_i)$
    add $\langle c^t{:}m_{this}, \{c'{:}o_i\}\rangle$ to *WL*
    if $c{:}l \rightarrow c^t{:}m \notin CG$ **then**
      add $c{:}l \rightarrow c^t{:}m$ to CG
      $\text{AddReachable}(c^t{:}m)$
      foreach parameter p_i of m **do**
        $\text{AddEdge}(c{:}a_i, c^t{:}p_i)$
      $\text{AddEdge}(c^t{:}m_{ret}, c{:}r)$

$m = \text{C.id(Number)}$
$c^t = [14]$

Processing: $\langle [4]{:}\text{C}.m_{this}, \{o_3\}\rangle$

PFG:

$\{o_3\}$ $[\,]{:}c$

$\{o_3\}$ $[4]{:}\text{C}.m_{this}$

$\text{ProcessCall}(c{:}x, \text{c}'{:}o_i)$

CG: $\{\ [\,]{:}4 \rightarrow [4]{:}\text{C.m()}\ \}$

```
1  class C {
2    static void main() {
3      C c = new C();
4      c.m();
5    }
6
7    Number id(Number n) {
8      return n;
9    }
10   void m() {
11     Number n1,n2,x,y;
12     n1 = new One();
13     n2 = new Two();
14     x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17   }
18 }
```

RM: $\{\ [\,]{:}\text{C.main()}, [4]{:}\text{C.m()}\ \}$

1-Call-Site: Example

WL: $[\langle [4]{:}n1, \{o_{12}\}\rangle, \langle [4]{:}n2, \{o_{13}\}\rangle]$

ProcessCall($c{:}\,x, c'{:}\,o_i$)
  foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
    m = Dispatch(o_i, k)
    c^t = Select($c, l, c'{:}\,o_i$)
    add $\langle c^t{:}\,m_{this}, \{c'{:}\,o_i\}\rangle$ to *WL*
    if $c{:}\,l \rightarrow c^t{:}\,m \notin CG$ **then**
      add $c{:}\,l \rightarrow c^t{:}\,m$ to *CG*
      AddReachable($c^t{:}\,m$)
      foreach parameter p_i of m **do**
        AddEdge($c{:}\,a_i,\ c^t{:}\,p_i$)
      AddEdge($c^t{:}\,m_{ret}, c{:}\,r$)

m = C.id(Number)
c^t = [14]

Processing: $\langle [4]{:}\mathrm{C}.m_{this}, \{o_3\}\rangle$

PFG:

$\{o_3\}$ [ ]:c

$\{o_3\}$ [4]:C.m_{this}

ProcessCall($c{:}\,x$, $c'{:}\,o_i$)

CG: { [ ]:4 → [4]:C.m(),
[4]:14 → [14]:C.id(Number) }

```
1  class C {
2    static void main() {
3      C c = new C();
4      c.m();
5    }
6
7    Number id(Number n) {
8      return n;
9    }
10   void m() {
11     Number n1,n2,x,y;
12     n1 = new One();
13     n2 = new Two();
14     x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17   }
18 }
```

RM: { [ ]:C.main(), [4]:C.m(),
[14]:C.id(Number) }

1-Call-Site: Example

WL: $[\langle [4]{:}n1, \{o_{12}\}\rangle, \langle [4]{:}n2, \{o_{13}\}\rangle]$

ProcessCall($c{:}x, c'{:}o_i$)
  foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
    m = Dispatch(o_i, k)
    c^t = Select($c, l, c'{:}o_i$)
    add $\langle c^t{:}m_{this}, \{c'{:}o_i\}\rangle$ to *WL*
    if $c{:}l \rightarrow c^t{:}m \notin CG$ **then**
      add $c{:}l \rightarrow c^t{:}m$ to CG
      AddReachable($c^t{:}m$)
      foreach parameter p_i of m **do**
        AddEdge($c{:}a_i, c^t{:}p_i$)
      AddEdge($c^t{:}m_{ret}, c{:}r$)

m = C.id(Number)
c^t = [14]

Processing: $\langle$[4]:C.m_{this}, $\{o_3\}\rangle$

PFG:

$\{o_3\}$ [ ]:c

$\{\}$ [4]:$n1$ → $\{\}$ [14]:n

$\{o_3\}$ [4]:C.m_{this}

ProcessCall($c{:}x$, $c'{:}o_i$)

```
1  class C {
2    static void main() {
3      C c = new C();
4      c.m();
5    }
6
7    Number id(Number n) {
8      return n;
9    }
10   void m() {
11     Number n1,n2,x,y;
12     n1 = new One();
13     n2 = new Two();
14     x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17   }
18 }
```

CG: { [ ]:4 → [4]:C.m(),
[4]:14 → [14]:C.id(Number) }

RM: { [ ]:C.main(), [4]:C.m(),
[14]:C.id(Number) }

1-Call-Site: Example

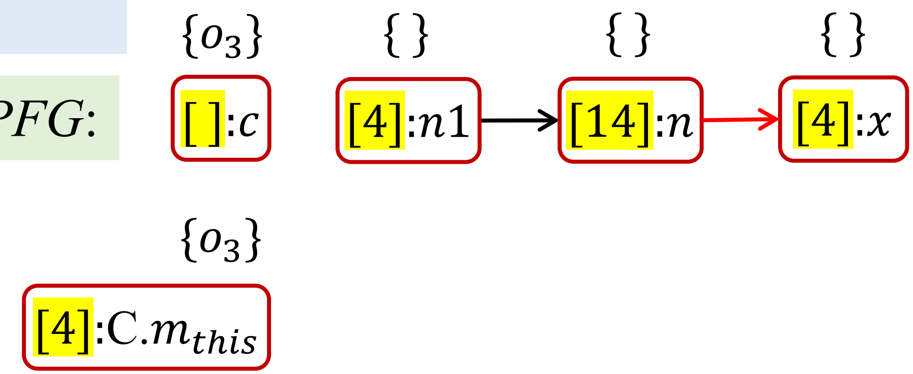
WL: [⟨[4]:$n1$, $\{o_{12}\}$⟩, ⟨[4]:$n2$, $\{o_{13}\}$⟩]

ProcessCall(c: x, c': o_i)
  foreach `l: r = x.k(a1,…,an)` ∈ S **do**
    m = Dispatch(o_i, k)
    c^t = Select(c, l, c': o_i)
    add ⟨c^t: m_{this}, $\{c'$: $o_i\}$⟩ to *WL*
    if c: $l \rightarrow c^t$: $m \notin CG$ **then**
      add c: $l \rightarrow c^t$: m to CG
      AddReachable(c^t: m)
      foreach parameter p_i of m **do**
        AddEdge(c: a_i, c^t: p_i)
      AddEdge(c^t: m_{ret}, c: r)

m = C.id(Number)
c^t = [14]

Processing: ⟨[4]:C.m_{this}, $\{o_3\}$⟩

PFG:



ProcessCall(c: x, c′: o_i)

CG: { [ ]:4 → [4]:C.m(), [4]:14 → [14]:C.id(Number) }

RM: { [ ]:C.main(), [4]:C.m(), [14]:C.id(Number) }

```
1  class C {
2    static void main() {
3      C c = new C();
4      c.m();
5    }
6
7    Number id(Number n) {
8      return n;
9    }
10   void m() {
11     Number n1,n2,x,y;
12     n1 = new One();
13     n2 = new Two();
14     x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17   }
18 }
```

1-Call-Site: Example

WL: $[\langle[4]{:}n1, \{o_{12}\}\rangle, \langle[4]{:}n2, \{o_{13}\}\rangle]$

ProcessCall($c{:}x, c'{:}o_i$)
  foreach l: `r = x.k(a1,…,an)` $\in S$ **do**
    m = Dispatch(o_i, k)
    c^t = Select($c, l, c'{:}o_i$)
    add $\langle c^t{:}m_{this}, \{c'{:}o_i\}\rangle$ to *WL*
    if $c{:}l \rightarrow c^t{:}m \notin CG$ **then**
      add $c{:}l \rightarrow c^t{:}m$ to CG
      AddReachable($c^t{:}m$)
      foreach parameter p_i of m **do**
        AddEdge($c{:}a_i$, $c^t{:}p_i$)
      AddEdge($c^t{:}m_{ret}, c{:}r$)

m = C.id(Number)
c^t = [15]

Processing: $\langle[4]{:}\text{C}.m_{this}, \{o_3\}\rangle$

PFG:

$\{o_3\}$ [ ]:c

$\{\}$ [4]:$n1$ → $\{\}$ [14]:n → $\{\}$ [4]:x

$\{o_3\}$ [4]:C.m_{this}

ProcessCall($c{:}x$, $c'{:}o_i$)

```
1 class C {                       10   void m() {
2   static void main() {          11     Number n1,n2,x,y;
3     C c = new C();              12     n1 = new One();
4     c.m();                      13     n2 = new Two();
5   }                             14     x = this.id(n1);
6                                 15 ⇒   y = this.id(n2);
7   Number id(Number n) {         16     x.get();
8     return n;                   17   }
9   }                             18 }
```

CG: { [ ]:4 → [4]:C.m(),
[4]:14 → [14]:C.id(Number),
[4]:15 → [15]:C.id(Number) }

RM: { [ ]:C.main(), [4]:C.m(),
[14]:C.id(Number),
[15]:C.id(Number) }

1-Call-Site: Example

WL: $[\langle [4]{:}n1, \{o_{12}\}\rangle, \langle [4]{:}n2, \{o_{13}\}\rangle]$

ProcessCall($c{:}x, c'{:}o_i$)
  foreach `l: r = x.k(a1,…,an)` $\in S$ **do**
    m = Dispatch(o_i, k)
    c^t = Select($c, l, c'{:}o_i$)
    add $\langle c^t{:}m_{this}, \{c'{:}o_i\}\rangle$ to *WL*
    if $c{:}l \rightarrow c^t{:}m \notin CG$ **then**
      add $c{:}l \rightarrow c^t{:}m$ to CG
      AddReachable($c^t{:}m$)
      foreach parameter p_i of m **do**
        AddEdge($c{:}a_i$, $c^t{:}p_i$)
      AddEdge($c^t{:}m_{ret}, c{:}r$)

m = C.id(Number)
c^t = [15]

Processing: $\langle [4]{:}C.m_{this}, \{o_3\}\rangle$

PFG:

$\{o_3\}$ [ ]:c $\quad$ $\{\}$ [4]:$n1$ → $\{\}$ [14]:n → $\{\}$ [4]:x

$\{o_3\}$ [4]:C.m_{this} $\quad$ $\{\}$ [4]:$n2$ → $\{\}$ [15]:n → $\{\}$ [4]:y

ProcessCall($c{:}x$, $c'{:}o_i$)

```
1  class C {                          10   void m() {
2    static void main() {             11     Number n1,n2,x,y;
3      C c = new C();                 12     n1 = new One();
4      c.m();                         13     n2 = new Two();
5    }                                14     x = this.id(n1);
6                                     15     y = this.id(n2);
7    Number id(Number n) {            16     x.get();
8      return n;                      17   }
9    }                                18 }
```

CG: { [ ]:4 → [4]:C.m(),
[4]:14 → [14]:C.id(Number),
[4]:15 → [15]:C.id(Number) }

RM: { [ ]:C.main(), [4]:C.m(),
[14]:C.id(Number),
[15]:C.id(Number) }

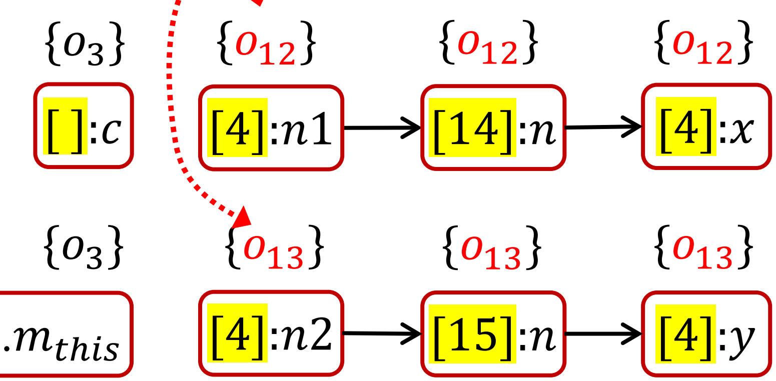
1-Call-Site: Example

WL: $[\langle[4]{:}n1, \{o_{12}\}\rangle, \langle[4]{:}n2, \{o_{13}\}\rangle]$

Solve(m^{entry})
- WL=[ ],PFG={},S={},RM={},CG={}
- AddReachable([ ]: m^{entry})
- **while** WL is not empty **do**
  - remove $\langle n, pts\rangle$ from WL
  - $\Delta = pts - pt(n)$
  - Propagate(n, Δ)
  - **if** n represents a variable $c{:}\,x$ **then**
    - **foreach** $c'{:}\,o_i \in \Delta$ **do**
      - **foreach** `x.f = y` $\in S$ **do**
        - AddEdge($c{:}\,y$, $c'{:}\,o_i.f$)
      - **foreach** `y = x.f` $\in S$ **do**
        - AddEdge($c'{:}\,o_i.f$, $c{:}\,y$)
      - ProcessCall($c{:}\,x$, $c'{:}\,o_i$)

Processing: …

PFG:



CG: { [ ]:4 → [4]:C.m(),
[4]:14 → [14]:C.id(Number),
[4]:15 → [15]:C.id(Number) }

RM: { [ ]:C.main(), [4]:C.m(),
[14]:C.id(Number),
[15]:C.id(Number) }

```
1  class C {
2    static void main() {
3      C c = new C();
4      c.m();
5    }
6
7    Number id(Number n) {
8      return n;
9    }
10   void m() {
11     Number n1,n2,x,y;
12     n1 = new One();
13     n2 = new Two();
14     x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17   }
18 }
```

1-Call-Site: Example

WL: [ ]

Solve(m^{entry})
  WL=[ ],*PFG*={},*S*={},*RM*={},*CG*={}
  AddReachable([ ]: m^{entry})
  while *WL* is not empty **do**
    remove $\langle n, pts\rangle$ from *WL*
    $\Delta = pts - pt(n)$
    Propagate(n, Δ)
    if n represents a variable c: x **then**
      foreach c': $o_i \in \Delta$ **do**
        foreach `x.f = y` $\in S$ **do**
          AddEdge(c: y, c': $o_i.f$)
        foreach `y = x.f` $\in S$ **do**
          AddEdge(c': $o_i.f$, c: y)
      ProcessCall(c: x, c': o_i)

Processing: $\langle [4]\!: x, \{o_{12}\}\rangle$

PFG:

- [ ]:c $\{o_3\}$
- [4]:$n1$ $\{o_{12}\}$ → [14]:n $\{o_{12}\}$ → [4]:x $\{o_{12}\}$
- [4]:C.m_{this} $\{o_3\}$
- [4]:$n2$ $\{o_{13}\}$ → [15]:n $\{o_{13}\}$ → [4]:y $\{o_{13}\}$

CG: { [ ]:4 → [4]:C.m(),
[4]:14 → [14]:C.id(Number),
[4]:15 → [15]:C.id(Number),
[4]:16 → [16]:One.get() }

RM: { [ ]:C.main(), [4]:C.m(),
[14]:C.id(Number),
[15]:C.id(Number),
[16]:One.get() }

```
1  class C {
2    static void main() {
3      C c = new C();
4      c.m();
5    }
6
7    Number id(Number n) {
8      return n;
9    }
10   void m() {
11     Number n1,n2,x,y;
12     n1 = new One();
13     n2 = new Two();
14     x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17   }
18 }
```

1-Call-Site: Example

WL: [ ]

$\text{Solve}(m^{entry})$
$WL=[\,], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$
$\text{AddReachable}([\,]{:}\, m^{entry})$
while *WL* is not empty **do**
  remove $\langle n, pts\rangle$ from *WL*
  $\Delta = pts - pt(n)$
  $\text{Propagate}(n, \Delta)$
  if n represents a variable $c{:}\,x$ **then**
    foreach $c'{:}\, o_i \in \Delta$ **do**
      foreach `x.f = y` $\in S$ **do**
        $\text{AddEdge}(c{:}\,y,\ c'{:}\,o_i.f)$
      foreach `y = x.f` $\in S$ **do**
        $\text{AddEdge}(c'{:}\,o_i.f,\ c{:}\,y)$
      $\text{ProcessCall}(c{:}\,x,\ c'{:}\,o_i)$

Algorithm finishes

Final results

PFG:

$[\,]{:}c$ $\{o_3\}$

$[4]{:}n1$ $\{o_{12}\}$ → $[14]{:}n$ $\{o_{12}\}$ → $[4]{:}x$ $\{o_{12}\}$

$[4]{:}C.m_{this}$ $\{o_3\}$

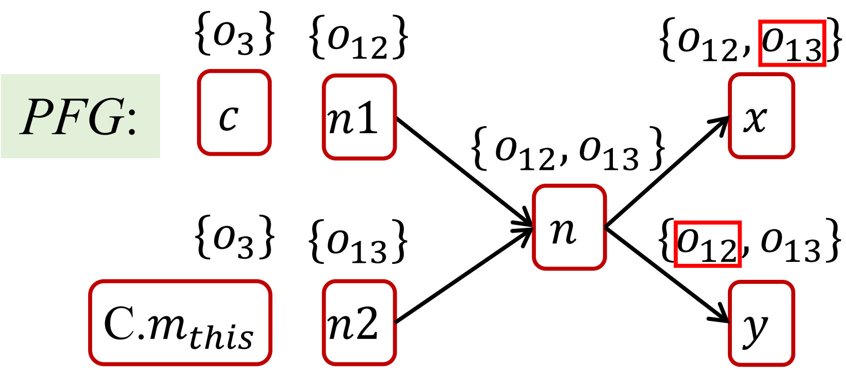
$[4]{:}n2$ $\{o_{13}\}$ → $[15]{:}n$ $\{o_{13}\}$ → $[4]{:}y$ $\{o_{13}\}$

CG: { [ ]:4 → [4]:C.m(),
[4]:14 → [14]:C.id(Number),
[4]:15 → [15]:C.id(Number),
[4]:16 → [16]:One.get() }

```
1  class C {
2    static void main() {
3      C c = new C();
4      c.m();
5    }
6
7    Number id(Number n) {
8      return n;
9    }
10   void m() {
11     Number n1,n2,x,y;
12     n1 = new One();
13     n2 = new Two();
14     x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17   }
18 }
```

RM: { [ ]:C.main(), [4]:C.m(),
[14]:C.id(Number),
[15]:C.id(Number),
[16]:One.get() }

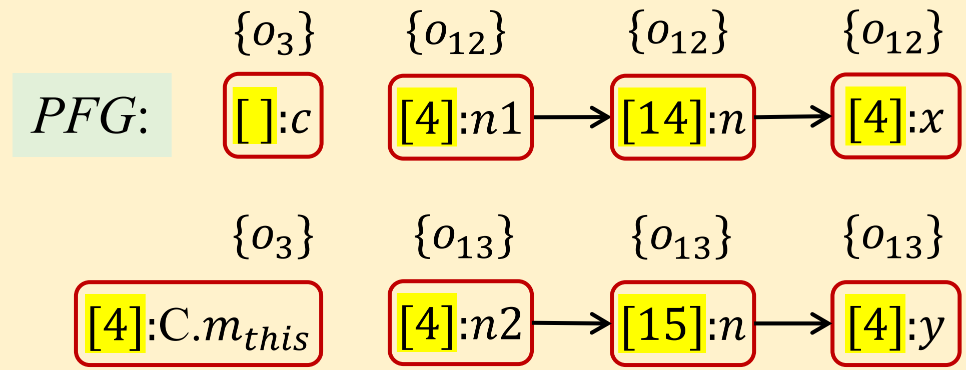
C.I. vs. C.S. (1-Call-Site)



CG: { 4 → C.m(),
14 → C.id(Number),
15 → C.id(Number),
16 → One.get(),
16 → Two.get() }

Spurious

Final results



CG: { [ ]:4 → [4]:C.m(),
[4]:14 → [14]:C.id(Number),
[4]:15 → [15]:C.id(Number),
[4]:16 → [16]:One.get() }

```
1  class C {
2    static void main() {
3      C c = new C();
4      c.m();
5    }
6
7    Number id(Number n) {
8      return n;
9    }
10   void m() {
11     Number n1,n2,x,y;
12     n1 = new One();
13     n2 = new Two();
14     x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17   }
18 }
```

Object Sensitivity*

- Each context consists of a list of abstract objects (represented by their allocation sites)
  - At a method call, use the receiver object with its heap context as callee context
  - Distinguish the operations of data flow on different objects

$$\mathrm{Select}(\_,\_,\boldsymbol{c}':\boldsymbol{o_i}) = [o_j, \dots, o_k, o_i]$$
$$\text{where } c' = [o_j, \dots, o_k]$$

Essentially "allocation-site sensitivity"

* Ana Milanova, Atanas Rountev, and Barbara G. Ryder. "*Parameterized Object Sensitivity for Points-to and Side-Effect Analyses for Java*". ISSTA 2002.

Object Sensitivity: Example

```
a1 = new A();
a2 = new A();
b1 = new B();
b2 = new B();
a1.set(b1);
a2.set(b2);
x = a1.get();

class A {
  B f;
  void set(B b) {
    this.doSet(b);
  }
  void doSet(B p) {
    this.f = p;
  }
  B get() {
    return this.f;
  }
}
```

Object Sensitivity: Example

```
a1 = new A(); $a1 \to o_1$
a2 = new A(); $a2 \to o_2$
b1 = new B(); $b1 \to o_3$
b2 = new B(); $b2 \to o_4$
a1.set(b1);
a2.set(b2);
x = a1.get();

class A {
  B f;
  void set(B b) {
    this.doSet(b);
  }
  void doSet(B p) {
    this.f = p;
  }
  B get() {
    return this.f;
  }
}
```

Object Sensitivity: Example

```
a1 = new A(); a1 → o1
a2 = new A(); a2 → o2
b1 = new B(); b1 → o3
b2 = new B(); b2 → o4
a1.set(b1);
a2.set(b2);
x = a1.get();

class A {
  B f;
  void set(B b) {
    this.doSet(b);
  }
  void doSet(B p) {
    this.f = p;
  }
  B get() {
    return this.f;
  }
}
```

$c^t = ?$

Object Sensitivity: Example

```
a1 = new A();
a2 = new A();
b1 = new B();
b2 = new B();
a1.set(b1);
a2.set(b2);
x = a1.get();

class A {
  B f;
  void set(B b) {
    this.doSet(b);
  }
  void doSet(B p) {
    this.f = p;
  }
  B get() {
    return this.f;
  }
}
```

$a1 \rightarrow o_1$

$a2 \rightarrow o_2$

$b1 \rightarrow o_3$

$b2 \rightarrow o_4$

$c^t = [o_1]$

Variable	Object
$[o_1]: set_{this}$	o_1
$[o_1]: b$	o_3

Object Sensitivity: Example

```
a1 = new A();   a1 → o1
a2 = new A();   a2 → o2
b1 = new B();   b1 → o3
b2 = new B();   b2 → o4
a1.set(b1);
a2.set(b2);
x = a1.get();

class A {
  B f;
  void set(B b) {
    this.doSet(b);
  }
  void doSet(B p) {
    this.f = p;
  }
  B get() {
    return this.f;
  }
}
```

$c^t = [o_1]$

$c^t =$?

Variable	Object
$[o_1]: set_{this}$	o_1
$[o_1]: b$	o_3

Object Sensitivity: Example

```
a1 = new A();
a2 = new A();
b1 = new B();
b2 = new B();
a1.set(b1);
a2.set(b2);
x = a1.get();

class A {
  B f;
  void set(B b) {
    this.doSet(b);
  }
  void doSet(B p) {
    this.f = p;
  }
  B get() {
    return this.f;
  }
}
```

$a1 \rightarrow o_1$

$a2 \rightarrow o_2$

$b1 \rightarrow o_3$

$b2 \rightarrow o_4$

$c^t = [o_1]$

$c^t = [o_1]$

Variable	Object
$[o_1]: set_{this}$	o_1
$[o_1]: b$	o_3
$[o_1]: doSet_{this}$	o_1
$[o_1]: p$	o_3

Field	Object
o_1.f	o_3

Object Sensitivity: Example

```
a1 = new A();   a1 → o1
a2 = new A();   a2 → o2
b1 = new B();   b1 → o3
b2 = new B();   b2 → o4
a1.set(b1);
a2.set(b2);
x = a1.get();

class A {
  B f;
  void set(B b) {
    this.doSet(b);
  }
  void doSet(B p) {
    this.f = p;
  }
  B get() {
    return this.f;
  }
}
```

$c^t = [o_1]$

$c^t = [o_2]$

$c^t = [o_1]$

$c^t = [o_2]$

Variable	Object
$[o_1]: set_{this}$	o_1
$[o_1]: b$	o_3
$[o_1]: doSet_{this}$	o_1
$[o_1]: p$	o_3
$[o_2]: set_{this}$	o_2
$[o_2]: b$	o_4
$[o_2]: doSet_{this}$	o_2
$[o_2]: p$	o_4

Field	Object
o_1.f	o_3
o_2.f	o_4

Object Sensitivity: Example

```
a1 = new A();
a2 = new A();
b1 = new B();
b2 = new B();
a1.set(b1);
a2.set(b2);
x = a1.get();

class A {
  B f;
  void set(B b) {
    this.doSet(b);
  }
  void doSet(B p) {
    this.f = p;
  }
  B get() {
    return this.f;
  }
}
```

$a1 \rightarrow o_1$

$a2 \rightarrow o_2$

$b1 \rightarrow o_3$

$b2 \rightarrow o_4$

$c^t = [o_1]$

$c^t = [o_2]$

$c^t = [o_1]$

$c^t = [o_2]$

Variable	Object
$[o_1]: set_{this}$	o_1
$[o_1]: b$	o_3
$[o_1]: doSet_{this}$	o_1
$[o_1]: p$	o_3
$[o_2]: set_{this}$	o_2
$[o_2]: b$	o_4
$[o_2]: doSet_{this}$	o_2
$[o_2]: p$	o_4
x	o_3

Field	Object
o_1.f	o_3
o_2.f	o_4

Call-Site vs. Object Sensitivity

1-call-site

1-object

```
a1 = new A(); a1 → o1
a2 = new A(); a2 → o2
b1 = new B(); b1 → o3
b2 = new B(); b2 → o4
a1.set(b1);
a2.set(b2);
x = a1.get();

class A {
  B f;
  void set(B b) {
    this.doSet(b);
  }
  void doSet(B p) {
    this.f = p;
  }
  B get() {
    return this.f;
  }
}
```

Variable	Object
$[o_1]: set_{this}$	o_1
$[o_1]: b$	o_3
$[o_1]: doSet_{this}$	o_1
$[o_1]: p$	o_3
$[o_2]: set_{this}$	o_2
$[o_2]: b$	o_4
$[o_2]: doSet_{this}$	o_2
$[o_2]: p$	o_4
x	o_3

Field	Object
o_1.f	o_3
o_2.f	o_4

Call-Site vs. Object Sensitivity

```
1  a1 = new A();   a1 → o1
2  a2 = new A();   a2 → o2
3  b1 = new B();   b1 → o3
4  b2 = new B();   b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10   B f;
11   void set(B b) {
12     this.doSet(b);
13   }
14   void doSet(B p) {
15     this.f = p;
16   }
17   B get() {
18     return this.f;
19   }
20 }
```

$c^t = [5]$

1-call-site

Variable	Object
$[5]$: set_{this}	o_1
$[5]$: b	o_3

1-object

Variable	Object
$[o_1]$: set_{this}	o_1
$[o_1]$: b	o_3
$[o_1]$: $doSet_{this}$	o_1
$[o_1]$: p	o_3
$[o_2]$: set_{this}	o_2
$[o_2]$: b	o_4
$[o_2]$: $doSet_{this}$	o_2
$[o_2]$: p	o_4
x	o_3
Field	**Object**
o_1.f	o_3
o_2.f	o_4

Call-Site vs. Object Sensitivity

```
1  a1 = new A();  a1 → o1
2  a2 = new A();  a2 → o2
3  b1 = new B();  b1 → o3
4  b2 = new B();  b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10   B f;
11   void set(B b) {
12     this.doSet(b);
13   }
14   void doSet(B p) {
15     this.f = p;
16   }
17   B get() {
18     return this.f;
19   }
20 }
```

$c^t = [5]$

$c^t = [12]$

1-call-site

Variable	Object
[5]: set_{this}	o_1
[5]: b	o_3
[12]: $doSet_{this}$	o_1
[12]: p	o_3
Field	**Object**
o_1.f	o_3

1-object

Variable	Object
$[o_1]$: set_{this}	o_1
$[o_1]$: b	o_3
$[o_1]$: $doSet_{this}$	o_1
$[o_1]$: p	o_3
$[o_2]$: set_{this}	o_2
$[o_2]$: b	o_4
$[o_2]$: $doSet_{this}$	o_2
$[o_2]$: p	o_4
x	o_3
Field	**Object**
o_1.f	o_3
o_2.f	o_4

Call-Site vs. Object Sensitivity

```
1  a1 = new A();  a1 → o1
2  a2 = new A();  a2 → o2
3  b1 = new B();  b1 → o3
4  b2 = new B();  b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10   B f;
11   void set(B b) {
12     this.doSet(b);
13   }
14   void doSet(B p) {
15     this.f = p;
16   }
17   B get() {
18     return this.f;
19   }
20 }
```

$c^t = [5]$

$c^t = [6]$

$c^t = [12]$

1-call-site

Variable	Object
[5]: set_{this}	o_1
[5]: b	o_3
[6]: set_{this}	o_2
[6]: b	o_4
[12]: $doSet_{this}$	o_1
[12]: p	o_3

Field	Object
o_1.f	o_3

1-object

Variable	Object
$[o_1]$: set_{this}	o_1
$[o_1]$: b	o_3
$[o_1]$: $doSet_{this}$	o_1
$[o_1]$: p	o_3
$[o_2]$: set_{this}	o_2
$[o_2]$: b	o_4
$[o_2]$: $doSet_{this}$	o_2
$[o_2]$: p	o_4
x	o_3

Field	Object
o_1.f	o_3
o_2.f	o_4

Call-Site vs. Object Sensitivity

```
1  a1 = new A();  a1 → o1
2  a2 = new A();  a2 → o2
3  b1 = new B();  b1 → o3
4  b2 = new B();  b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10   B f;
11   void set(B b) {
12     this.doSet(b);
13   }
14   void doSet(B p) {
15     this.f = p;
16   }
17   B get() {
18     return this.f;
19   }
20 }
```

$c^t = [5]$

$c^t = [6]$

$c^t = [12]$

$c^t =$?

1-call-site

Variable	Object
$[5]: set_{this}$	o_1
$[5]: b$	o_3
$[6]: set_{this}$	o_2
$[6]: b$	o_4
$[12]: doSet_{this}$	o_1
$[12]: p$	o_3

Field	Object
o_1.f	o_3

1-object

Variable	Object
$[o_1]: set_{this}$	o_1
$[o_1]: b$	o_3
$[o_1]: doSet_{this}$	o_1
$[o_1]: p$	o_3
$[o_2]: set_{this}$	o_2
$[o_2]: b$	o_4
$[o_2]: doSet_{this}$	o_2
$[o_2]: p$	o_4
x	o_3

Field	Object
o_1.f	o_3
o_2.f	o_4

Call-Site vs. Object Sensitivity

```
1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10   B f;
11   void set(B b) {
12     this.doSet(b);
13   }
14   void doSet(B p) {
15     this.f = p;
16   }
17   B get() {
18     return this.f;
19   }
20 }
```

$c^t = [5]$

$c^t = [6]$

$c^t = [12]$

1-call-site

Variable	Object
[5]: set_{this}	o_1
[5]: b	o_3
[6]: set_{this}	o_2
[6]: b	o_4
[12]: $doSet_{this}$	o_1, o_2
[12]: p	o_3, o_4
Field	**Object**
o_1.f	o_3, o_4
o_2.f	o_3, o_4

1-object

Variable	Object
[o_1]: set_{this}	o_1
[o_1]: b	o_3
[o_1]: $doSet_{this}$	o_1
[o_1]: p	o_3
[o_2]: set_{this}	o_2
[o_2]: b	o_4
[o_2]: $doSet_{this}$	o_2
[o_2]: p	o_4
x	o_3
Field	**Object**
o_1.f	o_3
o_2.f	o_4

Call-Site vs. Object Sensitivity

```
1   a1 = new A();   a1 → o1
2   a2 = new A();   a2 → o2
3   b1 = new B();   b1 → o3
4   b2 = new B();   b2 → o4
5   a1.set(b1);
6   a2.set(b2);
7   x = a1.get();
8
9   class A {
10    B f;
11    void set(B b) {
12      this.doSet(b);
13    }
14    void doSet(B p) {
15      this.f = p;
16    }
17    B get() {
18      return this.f;
19    }
20  }
```

$c^t = [5]$

$c^t = [6]$

$c^t = [12]$

1-call-site

Variable	Object
[5]: set_{this}	o_1
[5]: b	o_3
[6]: set_{this}	o_2
[6]: b	o_4
[12]: $doSet_{this}$	o_1, o_2
[12]: p	o_3, o_4
x	o_3, o_4
Field	**Object**
o_1.f	o_3, o_4
o_2.f	o_3, o_4

1-object

Variable	Object
[o_1]: set_{this}	o_1
[o_1]: b	o_3
[o_1]: $doSet_{this}$	o_1
[o_1]: p	o_3
[o_2]: set_{this}	o_2
[o_2]: b	o_4
[o_2]: $doSet_{this}$	o_2
[o_2]: p	o_4
x	o_3
Field	**Object**
o_1.f	o_3
o_2.f	o_4

Call-Site vs. Object Sensitivity

```
1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10   B f;
11   void set(B b) {
12     this.doSet(b);
13   }
14   void doSet(B p) {
15     this.f = p;
16   }
17   B get() {
18     return this.f;
19   }
20 }
```

$c^t = [5]$

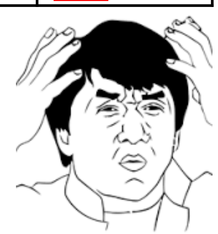
$c^t = [6]$

$c^t = [12]$

1-call-site

Variable	Object
[5]: set_{this}	o_1
[5]: b	o_3
[6]: set_{this}	o_2
[6]: b	o_4
[12]: $doSet_{this}$	o_1, o_2
[12]: p	o_3, o_4
x	$o_3, \boxed{o_4}$
Field	**Object**
o_1.f	$o_3, \boxed{o_4}$
o_2.f	$\boxed{o_3}, o_4$

Spurious

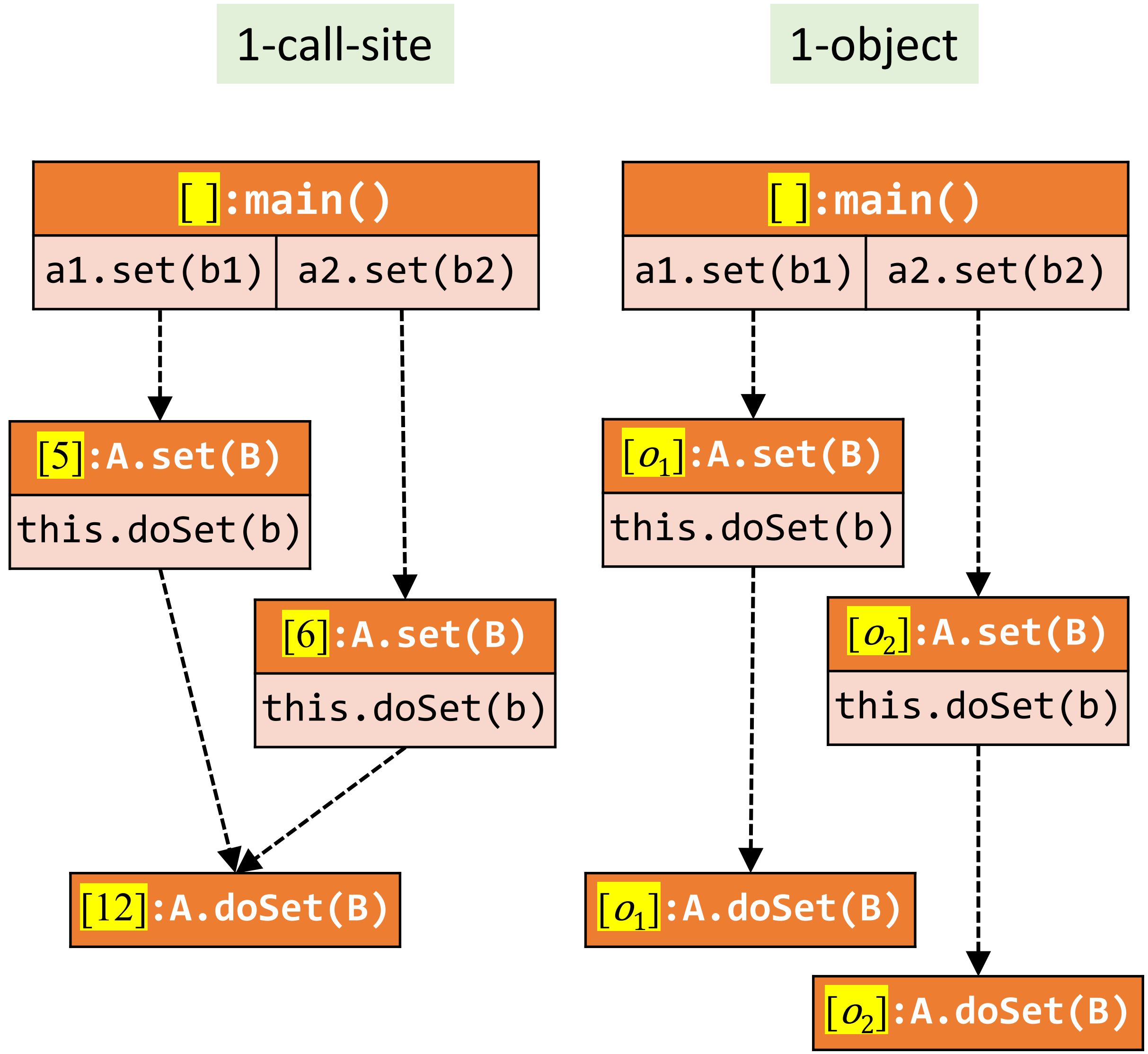


1-object

Variable	Object
[o_1]: set_{this}	o_1
[o_1]: b	o_3
[o_1]: $doSet_{this}$	o_1
[o_1]: p	o_3
[o_2]: set_{this}	o_2
[o_2]: b	o_4
[o_2]: $doSet_{this}$	o_2
[o_2]: p	o_4
x	o_3
Field	**Object**
o_1.f	o_3
o_2.f	o_4

Call-Site vs. Object Sensitivity

```
a1 = new A();
a2 = new A();
b1 = new B();
b2 = new B();
a1.set(b1);
a2.set(b2);
x = a1.get();

class A {
  B f;
  void set(B b) {
    this.doSet(b);
  }
  void doSet(B p) {
    this.f = p;
  }
  B get() {
    return this.f;
  }
}
```



C.S. (1-Object) vs. C.S. (1-Call-Site)

Final results

PFG:

$\{o_3\}$ [ ]:c

$\{o_{12}\}$ [4]:$n1$ $\rightarrow$ $\{o_{12}\}$ [14]:n $\rightarrow$ $\{o_{12}\}$ [4]:x

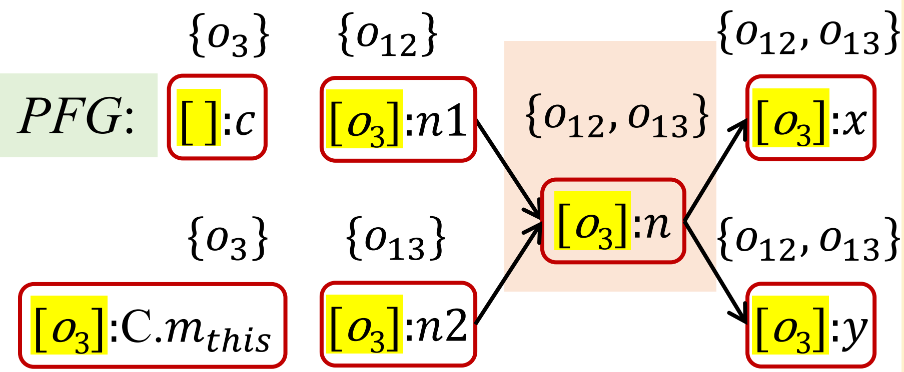
$\{o_3\}$ [4]:C.m_{this}

$\{o_{13}\}$ [4]:$n2$ $\rightarrow$ $\{o_{13}\}$ [15]:n $\rightarrow$ $\{o_{13}\}$ [4]:y

CG: { [ ]:4 → [4]:C.m(),
[4]:14 → [14]:C.id(Number),
[4]:15 → [15]:C.id(Number),
[4]:16 → [16]:One.get() }

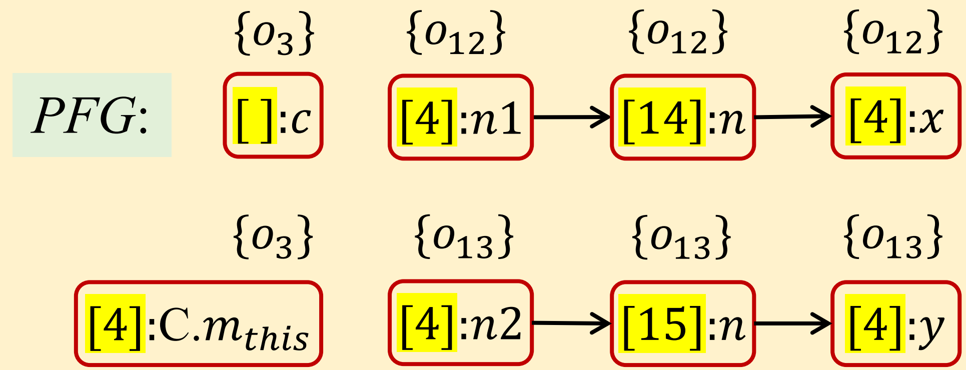
```
1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6
7   Number id(Number n) {
8     return n;
9   }
10  void m() {
11    Number n1,n2,x,y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get(); ?
17  }
18 }
```

C.S. (1-Object) vs. C.S. (1-Call-Site)



CG: { $[\,]$:4 → $[o_3]$:C.m(),
$[o_3]$:14 → $[o_3]$:C.id(Number),
$[o_3]$:15 → $[o_3]$:C.id(Number),
$[o_3]$:16 → $[o_{12}]$:One.get(),
$[o_3]$:16 → $[o_{13}]$:Two.get() }

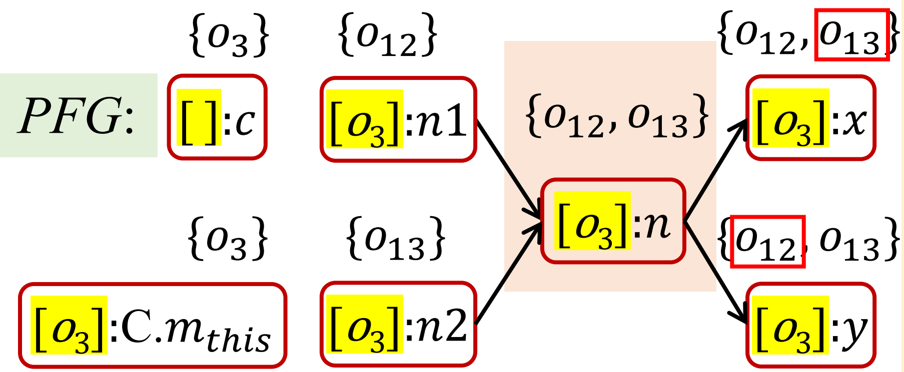
Final results



CG: { $[\,]$:4 → $[4]$:C.m(),
$[4]$:14 → $[14]$:C.id(Number),
$[4]$:15 → $[15]$:C.id(Number),
$[4]$:16 → $[16]$:One.get() }

```
1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6
7   Number id(Number n) {
8     return n;
9   }
10  void m() {
11    Number n1,n2,x,y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get(); ?
17  }
18 }
```

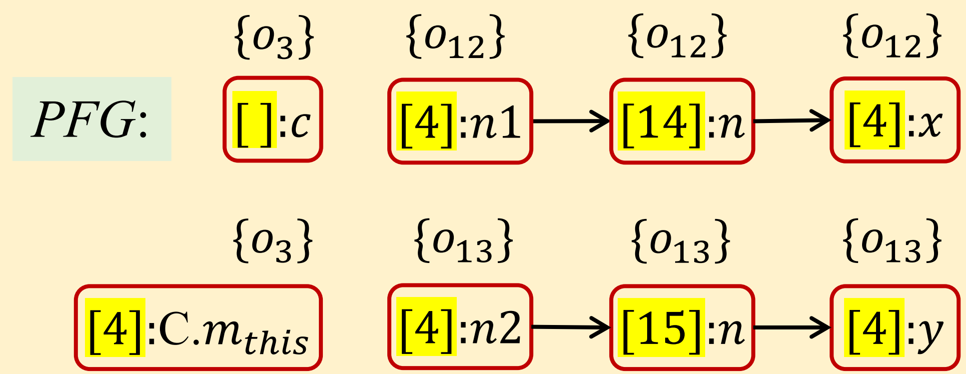
C.S. (1-Object) vs. C.S. (1-Call-Site)



CG: { [ ]:4 → $[o_3]$:C.m(),
$[o_3]$:14 → $[o_3]$:C.id(Number),
$[o_3]$:15 → $[o_3]$:C.id(Number),
$[o_3]$:16 → $[o_{12}]$:One.get(),
$[o_3]$:16 → $[o_{13}]$:Two.get() }

Spurious

Final results



CG: { [ ]:4 → [4]:C.m(),
[4]:14 → [14]:C.id(Number),
[4]:15 → [15]:C.id(Number),
[4]:16 → [16]:One.get() }

```
1 class C {                      10   void m() {
2   static void main() {         11     Number n1,n2,x,y;
3     C c = new C();             12     n1 = new One();
4     c.m();                     13     n2 = new Two();
5   }                            14     x = this.id(n1);
6                                15     y = this.id(n2);
7   Number id(Number n) {        16     x.get();
8     return n;                  17   }
9   }                            18 }
```

Call-Site vs. Object Sensitivity

- In theory, their precision is incomparable

Call-Site vs. Object Sensitivity

- In theory, their precision is incomparable

- In practice, object sensitivity generally outperforms call-site sensitivity for OO languages (like Java)

Call-Site vs. Object Sensitivity

	Time (s)		#may-fail-cast		#call-graph-edge	
	2-call	2-obj	2-call	2-obj	2-call	2-obj
batik	6,886	3,300	2,452	1,606	94,211	76,807
checkstyle	2,277	2,003	863	581	54,171	48,809
sunflow	5,570	1,208	2,504	1,837	100,701	89,866
findbugs	3,812	2,661	2,056	1,409	72,118	65,836
jpc	3,343	559	1,855	1,392	89,677	81,030
eclipse	1,896	146	886	546	42,872	38,151
chart	2,705	282	1,481	883	59,691	52,374
fop	5,503	1,200	1,975	1,446	79,524	71,408
xalan	1,927	1,093	919	533	48,763	44,871
bloat	5,712	3,525	1,699	1,193	58,696	53,143

For all numbers, lower is better (in terms of efficiency or precision)

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. "*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*". TOPLAS 2020.

Call-Site vs. Object Sensitivity

	Time (s)		#may-fail-cast		#call-graph-edge	
	2-call	2-obj	2-call	2-obj	2-call	2-obj
batik	6,886	3,300	2,452	1,606	94,211	76,807
checkstyle	2,277	2,003	863	581	54,171	48,809
sunflow	5,570	1,208	2,504	1,837	100,701	89,866
findbugs	3,812	2,661	2,056	1,409	72,118	65,836
jpc	3,343	559	1,855	1,392	89,677	81,030
eclipse	1,896	146	886	546	42,872	38,151
chart	2,705	282	1,481	883	59,691	52,374
fop	5,503	1,200	1,975	1,446	79,524	71,408
xalan	1,927	1,093	919	533	48,763	44,871
bloat	5,712	3,525	1,699	1,193	58,696	53,143

```
A a = (A) o;
```

For all numbers, lower is better (in terms of efficiency or precision)

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. “*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*”. TOPLAS 2020.

Call-Site vs. Object Sensitivity

	Time (s)		#may-fail-cast		#call-graph-edge	
	2-call	2-obj	2-call	2-obj	2-call	2-obj
batik	6,886	3,300	2,452	1,606	94,211	76,807
checkstyle	2,277	2,003	863	581	54,171	48,809
sunflow	5,570	1,208	2,504	1,837	100,701	89,866
findbugs	3,812	2,661	2,056	1,409	72,118	65,836
jpc	3,343	559	1,855	1,392	89,677	81,030
eclipse	1,896	146	886	546	42,872	38,151
chart	2,705	282	1,481	883	59,691	52,374
fop	5,503	1,200	1,975	1,446	79,524	71,408
xalan	1,927	1,093	919	533	48,763	44,871
bloat	5,712	3,525	1,699	1,193	58,696	53,143

For all numbers, lower is better (in terms of eff

In general
- Precision: object > call-site
- Efficiency: object > call-site

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. "*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*". TOPLAS 2020.

Type Sensitivity*

- Each context consists of a list of types
  - At a method call, use the type containing the allocation site of the receiver object with its heap context as callee context
  - A coarser abstraction over object sensitivity

$$\text{Select}(\_, \_, \boldsymbol{c'}\text{:}\,\boldsymbol{o_i}) = [t', \dots, t'', \text{InType}(o_i)]$$
$$\text{where } c' = [t', \dots, t'']$$

* Yannis Smaragdakis, Martin Bravenboer, and Ondrej Lhoták. "*Pick Your Contexts Well: Understanding Object-Sensitivity*". POPL 2011.

Type Sensitivity*

- Each context consists of a list of types
  - At a method call, use the type containing the allocation site of the receiver object with its heap context as callee context
  - A coarser abstraction over object sensitivity

$$\text{Select}(\_, \_, \boldsymbol{c}' {:} \boldsymbol{o_i}) = [t', \dots, t'', \text{InType}(o_i)]$$
$$\text{where } c' = [t', \dots, t'']$$

```
class X {
  void m() {
    Y y = new Y();
  }
}
```

$\text{InType}(o_3) = \text{X}$ (**not** Y!)

* Yannis Smaragdakis, Martin Bravenboer, and Ondrej Lhoták. "*Pick Your Contexts Well: Understanding Object-Sensitivity*". POPL 2011.

Type vs. Object Sensitivity

Under the same *k*-limiting, the precision of type sensitivity is no better than object sensitivity

- Type sensitivity is a coarser abstraction over object sensitivity

```
class X {
  void main() {
    Y y1 = new Y();
    y1.foo();
    Y y2 = new Y();
    y2.foo();
    Y y3 = new Y();
    y3.foo();
  }
}
class Y {
  void foo() {}
}
```

Contexts of `foo()`

➢ Object-sensitivity: ?

Type vs. Object Sensitivity

Under the same *k*-limiting, the precision of type sensitivity is no better than object sensitivity

- Type sensitivity is a coarser abstraction over object sensitivity

```
1  class X {
2    void main() {
3      Y y1 = new Y();
4      y1.foo();
5      Y y2 = new Y();
6      y2.foo();
7      Y y3 = new Y();
8      y3.foo();
9    }
10 }
11 class Y {
12   void foo() {}
13 }
```

Contexts of `foo()`

- Object-sensitivity: $[o_3]$, $[o_5]$, $[o_7]$
- Type-sensitivity: ?

Type vs. Object Sensitivity

Under the same *k*-limiting, the precision of type sensitivity is no better than object sensitivity

- Type sensitivity is a coarser abstraction over object sensitivity

```
class X {
  void main() {
    Y y1 = new Y();
    y1.foo();
    Y y2 = new Y();
    y2.foo();
    Y y3 = new Y();
    y3.foo();
  }
}
class Y {
  void foo() {}
}
```

Contexts of `foo()`

- Object-sensitivity: $[o_3], [o_5], [o_7]$
- Type-sensitivity: $[X]$

Type vs. Object Sensitivity

Under the same *k*-limiting, the precision of type sensitivity is no better than object sensitivity

- Type sensitivity is a coarser abstraction over object sensitivity

```
class X {
  void main() {
    Y y1 = new Y();
    y1.foo();
    Y y2 = new Y();
    y2.foo();
    Y y3 = new Y();
    y3.foo();
  }
}
class Y {
  void foo() {}
}
```

Compared to object sensitivity, type sensitivity trades precision for better efficiency, by merging the allocation sites in the same type in contexts

- In practice, type sensitivity is less precise but more efficient than object sensitivity

Contexts of `foo()`

- Object-sensitivity: $[o_3]$, $[o_5]$, $[o_7]$
- Type-sensitivity: $[X]$

Call-Site vs. Object vs. Type Sensitivity

	Time (s)			#may-fail-cast			#call-graph-edge		
	2-call	2-obj	2-type	2-call	2-obj	2-type	2-call	2-obj	2-type
batik	6,886	3,300	378	2,452	1,606	1,938	94,211	76,807	77,337
checkstyle	2,277	2,003	125	863	581	695	54,171	48,809	49,274
sunflow	5,570	1,208	197	2,504	1,837	2,247	100,701	89,866	90,967
findbugs	3,812	2,661	265	2,056	1,409	1,683	72,118	65,836	66,443
jpc	3,343	559	128	1,855	1,392	1,599	89,677	81,030	81,527
eclipse	1,896	146	57	886	546	665	42,872	38,151	38,337
chart	2,705	282	84	1,481	883	1,155	59,691	52,374	52,965
fop	5,503	1,200	251	1,975	1,446	1,753	79,524	71,408	71,847
xalan	1,927	1,093	99	919	533	729	48,763	44,871	45,444
bloat	5,712	3,525	74	1,699	1,193	1,486	58,696	53,143	54,279

For all numbers, lower is better (in terms of efficiency or precision)

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. "*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*". TOPLAS 2020.

Call-Site vs. Object vs. Type Sensitivity

	Time (s)			#may-fail-cast			#call-graph-edge		
	2-call	2-obj	2-type	2-call	2-obj	2-type	2-call	2-obj	2-type
batik	6,886	3,300	378	2,452	1,606	1,938	94,211	76,807	77,337
checkstyle	2,277	2,003	125	863	581	695	54,171	48,809	49,274
sunflow	5,570	1,208	197	2,504	1,837	2,247	100,701	89,866	90,967
findbugs	3,812	2,661	265	2,056	1,409	1,683	72,118	65,836	66,443
jpc	3,343	559	128	1,855	1,392	1,599	89,677	81,030	81,527
eclipse	1,896	146	57	886	546	665	42,872	38,151	38,337
chart	2,705	282	84	1,481	883	1,155	59,691	52,374	52,965
fop	5,503	1,200	251	1,975	1,446	1,753	79,524	71,408	71,847
xalan	1,927	1,093	99	919	533	729	48,763	44,871	45,444
bloat	5,712	3,525	74	1,699	1,193	1,486	58,696	53,143	54,279

For all numbers, lower is better (in terms

In general
- Precision: object > type > call-site
- Efficiency: type > object > call-site

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. "*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*". TOPLAS 2020.

The X You Need To Understand in This Lecture

- Algorithm for context-sensitive pointer analysis
- Common context sensitivity variants
- Differences and relationship among common context sensitivity variants

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