

# Constraint based analysis for Java

PAS/SDL – 2010/2011

David Pichardie

# Outline

- 1 Constraint based analysis
- 2 Static analysis for Java
- 3 Soundness proof of the points-to analysis

# Inequation systems

Analyses are sometimes specified as inequation (constraints) systems :

$$\begin{cases} x_1 \sqsupseteq f_1(x_1, \dots, x_n) \\ \vdots \\ x_n \sqsupseteq f_n(x_1, \dots, x_n) \end{cases}$$

**Knaster-Tarski** : For a complete lattice  $(A, \sqsupseteq, \sqsubseteq)$  and a monotone function  $f$

$$\mathbf{lfp}(f) = \bigsqcap \{x \in A \mid f(x) \sqsubseteq x\}$$

**Corollary** : Under the same hypothesis,  $\mathbf{lfp}(f) = \mathbf{lfpf}(f)$  where  $\mathbf{lfpf}(f)$  is the least post-fixpoint of  $f$ .

**Conclusion** : the two specification styles are equivalent.

# Data flow equations of reachable definitions

Previous lecture

Domain where the solution lives :  $RD_{in}(l), RD_{out}(l) \in \wp(Var \times Lab^?)$

$$kill([\mathbf{x} := a]^l) = \{(\mathbf{x}, l') \mid l' \in Lab^?\}$$

$$kill([\mathbf{skip}]^l) = \emptyset$$

$$kill([b]^l) = \emptyset$$

$$gen([\mathbf{x} := a]^l) = \{(x, l)\}$$

$$gen([\mathbf{skip}]^l) = \emptyset$$

$$gen([b]^l) = \emptyset$$

For all  $[b]^l \in P$ ,

$$RD_{in}(l) = \begin{cases} \{(\mathbf{x}, ?) \mid \mathbf{x} \in Var\} & \text{if } l = \mathit{init}(P) \\ \bigcup_{(l', l) \in \mathit{flow}(P)} RD_{out}(l') & \end{cases}$$

$$RD_{out}(l) = RD_{in}(l) \setminus kill([b]^l) \cup gen([b]^l)$$

# A constraint based specification

For all  $[b]^l \in P$ ,

$$\begin{aligned}
 RD_{\text{in}}(\mathit{init}(P)) &\supseteq \{(\mathbf{x}, ?) \mid \mathbf{x} \in \mathit{Var}\} \\
 RD_{\text{in}}(l) &\supseteq \bigcup_{(l', l) \in \mathit{flow}(P)} RD_{\text{out}}(l') \text{ if } l \neq \mathit{init}(P) \\
 RD_{\text{out}}(l) &\supseteq RD_{\text{in}}(l) \setminus \mathit{kill}([b]^l) \cup \mathit{gen}([b]^l)
 \end{aligned}$$

or

$$\begin{aligned}
 RD_{\text{in}}(\mathit{init}(P)) &\supseteq \{(\mathbf{x}, ?) \mid \mathbf{x} \in \mathit{Var}\} \\
 RD_{\text{in}}(l) &\supseteq RD_{\text{out}}(l') \quad \forall (l', l) \in \mathit{flow}(P), l \neq \mathit{init}(P) \\
 RD_{\text{out}}(l) &\supseteq RD_{\text{in}}(l) \setminus \mathit{kill}([b]^l) \cup \mathit{gen}([b]^l)
 \end{aligned}$$

This is still a *dataflow* style. We will now see a *syntax-directed* specification.

# A constraint based specification

We add a new label  $end(P)$  to each program  $P$ .

We attach an information  $RD(l) \in \wp(Var \times Lab^?)$  at each label of the program.

$RD$  is specified as the least<sup>1</sup> solution of the predicate  $RD \vdash P$  defined by

$$RD \vdash P \quad \text{iff} \quad \begin{array}{l} \{(x, ?) \mid x \in Var\} \subseteq RD(\mathit{init}(P)) \\ \text{and } RD \vdash (P, \mathit{end}(P)) \end{array}$$

where  $RD \vdash (S, l_{\text{next}})$  is defined for  $S \in \mathbf{Stm}$  and  $l_{\text{next}} \in \mathbf{Lab}$  by induction on  $\mathbf{Stm}$ .

Intuitively,  $RD \vdash (S, l_{\text{next}})$  constraints  $RD$  for all labels in statement  $S$ , plus the label  $l_{\text{next}}$  that represents the next label after  $S$ .

---

1. Does it make sense?

# A constraint based specification

$$\frac{RD(l) \setminus \{(x, l') \mid l' \in \text{Lab}^?\} \cup \{(x, l)\} \subseteq RD(l')}{RD \vdash ([x := a]^l, l')}$$

$$\frac{RD(l) \subseteq RD(l') \quad RD \vdash (S_1, \text{init}(S_2)) \quad RD \vdash (S_2, l')}{RD \vdash ([\text{skip}]^l, l') \quad RD \vdash (S_1 ; S_2, l')}$$

$$\frac{RD(l) \subseteq RD(\text{init}(S_1)) \quad RD \vdash (S_1, l') \quad RD(l) \subseteq RD(\text{init}(S_2)) \quad RD \vdash (S_2, l')}{RD \vdash (\text{if } [b]^l \text{ then } S_1 \text{ else } S_2, l')}$$

$$\frac{RD(l) \subseteq RD(\text{init}(S)) \quad RD \vdash (S, l) \quad RD(l) \subseteq RD(l')}{RD \vdash (\text{while } [b]^l \text{ do } S, l')}$$

# Example

$P = \text{if}[x = 0]^1 \text{ then } [\text{skip}]^2 \text{ else while } [y > 0]^3 \text{ do } ([x := x + 1]^4 ; [y := y - 1]^5)$

## Dataflow specification

$$RD_{\text{in}}(1) = \{(x, ?), (y, ?)\}$$

$$RD_{\text{in}}(2) = RD_{\text{out}}(1)$$

$$RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5)$$

$$RD_{\text{in}}(4) = RD_{\text{out}}(3)$$

$$RD_{\text{in}}(5) = RD_{\text{out}}(4)$$

$$RD_{\text{out}}(1) = RD_{\text{in}}(1)$$

$$RD_{\text{out}}(2) = RD_{\text{in}}(2)$$

$$RD_{\text{out}}(3) = RD_{\text{in}}(3)$$

$$RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(x, l') \mid l' \in \text{Lab}^?\} \cup \{(x, 4)\}$$

$$RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y, l') \mid l' \in \text{Lab}^?\} \cup \{(y, 5)\}$$

Constraint based, syntax-directed specification :

$$RD \vdash P$$

# Example

$$P = \text{if}[x = 0]^1 \text{ then } [\text{skip}]^2 \text{ else while } [y > 0]^3 \text{ do } ([x := x + 1]^4 ; [y := y - 1]^5)$$

## Dataflow specification

$$RD_{\text{in}}(1) = \{(x, ?), (y, ?)\}$$

$$RD_{\text{out}}(1) = RD_{\text{in}}(1)$$

$$RD_{\text{in}}(2) = RD_{\text{out}}(1)$$

$$RD_{\text{out}}(2) = RD_{\text{in}}(2)$$

$$RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5)$$

$$RD_{\text{out}}(3) = RD_{\text{in}}(3)$$

$$RD_{\text{in}}(4) = RD_{\text{out}}(3)$$

$$RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(x, l') \mid l' \in \text{Lab}^?\} \cup \{(x, 4)\}$$

$$RD_{\text{in}}(5) = RD_{\text{out}}(4)$$

$$RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y, l') \mid l' \in \text{Lab}^?\} \cup \{(y, 5)\}$$

Constraint based, syntax-directed specification :

$$\{(x, ?), (y, ?)\} \subseteq RD(1),$$

$$RD \vdash (\text{if}[x = 0]^1 \text{ then } [\text{skip}]^2 \text{ else while } [y > 0]^3 \text{ do } ([x := x + 1]^4 ; [y := y - 1]^5), 6)$$

# Example

$$P = \text{if}[x = 0]^1 \text{ then } [\text{skip}]^2 \text{ else while } [y > 0]^3 \text{ do } ([x := x + 1]^4 ; [y := y - 1]^5)$$

## Dataflow specification

$$RD_{\text{in}}(1) = \{(x, ?), (y, ?)\}$$

$$RD_{\text{in}}(2) = RD_{\text{out}}(1)$$

$$RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5)$$

$$RD_{\text{in}}(4) = RD_{\text{out}}(3)$$

$$RD_{\text{in}}(5) = RD_{\text{out}}(4)$$

$$RD_{\text{out}}(1) = RD_{\text{in}}(1)$$

$$RD_{\text{out}}(2) = RD_{\text{in}}(2)$$

$$RD_{\text{out}}(3) = RD_{\text{in}}(3)$$

$$RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(x, l') \mid l' \in \text{Lab}^?\} \cup \{(x, 4)\}$$

$$RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y, l') \mid l' \in \text{Lab}^?\} \cup \{(y, 5)\}$$

Constraint based, syntax-directed specification :

$$\{(x, ?), (y, ?)\} \subseteq RD(1),$$

$$RD(1) \subseteq RD(2), RD(1) \subseteq RD(3),$$

$$RD \vdash ([\text{skip}]^2, 6),$$

$$RD \vdash (\text{while } [y > 0]^3 \text{ do } ([x := x + 1]^4 ; [y := y - 1]^5), 6)$$

# Example

$P = \text{if}[x = 0]^1 \text{ then } [\text{skip}]^2 \text{ else while } [y > 0]^3 \text{ do } ([x := x + 1]^4 ; [y := y - 1]^5)$

## Dataflow specification

$$RD_{\text{in}}(1) = \{(x, ?), (y, ?)\}$$

$$RD_{\text{in}}(2) = RD_{\text{out}}(1)$$

$$RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5)$$

$$RD_{\text{in}}(4) = RD_{\text{out}}(3)$$

$$RD_{\text{in}}(5) = RD_{\text{out}}(4)$$

$$RD_{\text{out}}(1) = RD_{\text{in}}(1)$$

$$RD_{\text{out}}(2) = RD_{\text{in}}(2)$$

$$RD_{\text{out}}(3) = RD_{\text{in}}(3)$$

$$RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(x, l') \mid l' \in \text{Lab}^?\} \cup \{(x, 4)\}$$

$$RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y, l') \mid l' \in \text{Lab}^?\} \cup \{(y, 5)\}$$

## Constraint based, syntax-directed specification :

$$\{(x, ?), (y, ?)\} \subseteq RD(1),$$

$$RD(1) \subseteq RD(2), RD(1) \subseteq RD(3),$$

$$RD(2) \subseteq RD(6),$$

$$RD(3) \subseteq RD(4), RD(3) \subseteq RD(6),$$

$$RD \vdash ([x := x + 1]^4 ; [y := y - 1]^5, 3)$$

# Example

$P = \text{if}[x = 0]^1 \text{ then } [\text{skip}]^2 \text{ else while } [y > 0]^3 \text{ do } ([x := x + 1]^4 ; [y := y - 1]^5)$

## Dataflow specification

$$RD_{\text{in}}(1) = \{(x, ?), (y, ?)\}$$

$$RD_{\text{out}}(1) = RD_{\text{in}}(1)$$

$$RD_{\text{in}}(2) = RD_{\text{out}}(1)$$

$$RD_{\text{out}}(2) = RD_{\text{in}}(2)$$

$$RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5)$$

$$RD_{\text{out}}(3) = RD_{\text{in}}(3)$$

$$RD_{\text{in}}(4) = RD_{\text{out}}(3)$$

$$RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(x, l') \mid l' \in \text{Lab}^?\} \cup \{(x, 4)\}$$

$$RD_{\text{in}}(5) = RD_{\text{out}}(4)$$

$$RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y, l') \mid l' \in \text{Lab}^?\} \cup \{(y, 5)\}$$

## Constraint based, syntax-directed specification :

$$\{(x, ?), (y, ?)\} \subseteq RD(1),$$

$$RD(1) \subseteq RD(2), RD(1) \subseteq RD(3),$$

$$RD(2) \subseteq RD(6),$$

$$RD(3) \subseteq RD(4), RD(3) \subseteq RD(6),$$

$$RD \vdash ([x := x + 1]^4, 5),$$

$$RD \vdash ([y := y - 1]^5, 3)$$

# Example

$P = \text{if}[x = 0]^1 \text{ then } [\text{skip}]^2 \text{ else while } [y > 0]^3 \text{ do } ([x := x + 1]^4 ; [y := y - 1]^5)$

## Dataflow specification

$$RD_{\text{in}}(1) = \{(x, ?), (y, ?)\}$$

$$RD_{\text{out}}(1) = RD_{\text{in}}(1)$$

$$RD_{\text{in}}(2) = RD_{\text{out}}(1)$$

$$RD_{\text{out}}(2) = RD_{\text{in}}(2)$$

$$RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5)$$

$$RD_{\text{out}}(3) = RD_{\text{in}}(3)$$

$$RD_{\text{in}}(4) = RD_{\text{out}}(3)$$

$$RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(x, l') \mid l' \in \text{Lab}^?\} \cup \{(x, 4)\}$$

$$RD_{\text{in}}(5) = RD_{\text{out}}(4)$$

$$RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y, l') \mid l' \in \text{Lab}^?\} \cup \{(y, 5)\}$$

## Constraint based, syntax-directed specification :

$$\{(x, ?), (y, ?)\} \subseteq RD(1),$$

$$RD(1) \subseteq RD(2), RD(1) \subseteq RD(3),$$

$$RD(2) \subseteq RD(6),$$

$$RD(3) \subseteq RD(4), RD(3) \subseteq RD(6),$$

$$RD(4) \setminus \{(x, l') \mid l' \in \text{Lab}^?\} \cup \{(x, 4)\} \subseteq RD(5),$$

$$RD(5) \setminus \{(y, l') \mid l' \in \text{Lab}^?\} \cup \{(y, 5)\} \subseteq RD(3)$$

## Example : after simplification

$$P = \text{if } [x = 0]^1 \text{ then } [\text{skip}]^2 \text{ else while } [y > 0]^3 \text{ do } ([x := x + 1]^4 ; [y := y - 1]^5)$$

### Dataflow specification

$$RD_{\text{in}}(1) = \{(x, ?), (y, ?)\}$$

$$RD_{\text{in}}(2) = RD_{\text{in}}(1)$$

$$RD_{\text{in}}(3) = RD_{\text{in}}(1) \cup \left( RD_{\text{in}}(5) \setminus \{(y, l') \mid l' \in \text{Lab}^?\} \cup \{(y, 5)\} \right)$$

$$RD_{\text{in}}(4) = RD_{\text{in}}(3)$$

$$RD_{\text{in}}(5) = RD_{\text{in}}(4) \setminus \{(x, l') \mid l' \in \text{Lab}^?\} \cup \{(x, 4)\}$$

### Constraint based, syntax-directed specification :

$$RD(1) \supseteq \{(x, ?), (y, ?)\}$$

$$RD(2) \supseteq RD(1)$$

$$RD(3) \supseteq RD(1) \cup \left( RD(5) \setminus \{(y, l') \mid l' \in \text{Lab}^?\} \cup \{(y, 5)\} \right)$$

$$RD(4) \supseteq RD(3)$$

$$RD(5) \supseteq RD(4) \setminus \{(x, l') \mid l' \in \text{Lab}^?\} \cup \{(x, 4)\}$$

$$RD(6) \supseteq RD(2) \cup RD(3)$$

# Why a least solution ?

Because  $RD \vdash P$  is equivalent to

$$F_P(RD) \sqsubseteq RD$$

- ▶ in the canonic complete lattice on  $(\mathbf{Lab} \rightarrow \wp(\mathbf{Var} \times \mathbf{Lab}^?), \sqsubseteq, \sqcup)$ ,
- ▶ with  $F_P$  a monotone function.

# Discussion

## Constraint based specifications

- ▶ are more general than dataflow specifications (*i.e.* allow to formalize more static analyses)
- ▶ are often more easy to manipulate for soundness proofs
- ▶ generate constraint systems that may need to be « massaged » before being efficiently solved

# Outline

- 1 Constraint based analysis
- 2 Static analysis for Java
- 3 Soundness proof of the points-to analysis

# Analysing Java...

... is more difficult than analysing the *While* language!

- ▶ The control flow graph is more difficult to build (virtual calls, exceptions, ...).
- ▶ Memory updates are not so explicit because of aliasing :
  - ▶ example :

$y.f := 1 ; x.f := 0$

may end with  $y.f = 0!$

- ▶ Not to mention multi-threading, class initialisers, reflection, ...

and things get worse with C...

- ▶ no type safety, & operator, unions, formal semantics ?

# The $While_0$ language

We consider a toy object oriented language

## Syntax

- ▶ Variables ( $x, y \in \mathbf{Var}$ )
- ▶ Fields ( $f \in \mathbf{Field}$ )
- ▶ Classes ( $C \in \mathbf{Class}$ )
- ▶ Commands ( $S \in \mathbf{Stm}$ )

$$S ::= x := y \mid x := y.f \mid x.f := y \mid x := \mathbf{new} C \mid x := \mathbf{null} \\ \mid \mathbf{skip} \mid S_1 ; S_2 \mid \mathbf{if} (*) \mathbf{then} S_1 \mathbf{else} S_2 \mid \mathbf{while} (*) \mathbf{do} S$$

# Semantic domains

We assume a countable set **Ref** of references (heap address) and reserve the symbol  $\diamond$  for the **null** value.

- ▶ value : **Value** = **Ref** +  $\diamond$
- ▶ Local variables :  $\rho \in \mathbf{Loc} = \mathbf{Var} \rightarrow \mathbf{Value}$
- ▶ Heap :  $\sigma \in \mathbf{Heap} = \mathbf{Ref} \rightarrow (\mathbf{Field} \rightarrow \mathbf{Value})$  (partial function)
- ▶ State :  $(\rho, \sigma) \in \mathbf{Loc} \times \mathbf{Heap}$

We give to this language a (non deterministic) structural operational semantics.

## Structural operational semantics (1/2)

$$\frac{}{(x := \mathbf{null}, \rho, \sigma) \rightarrow \rho[\quad], \sigma}$$

$$\frac{}{(x := y, \rho, \sigma) \rightarrow \rho[\quad], \sigma}$$

$$\frac{\rho(y) \in \mathbf{dom}(\sigma)}{(x := y.f, \rho, \sigma) \rightarrow \rho[x \mapsto \quad], \sigma}$$

$$\frac{}{(x.f := y, \rho, \sigma) \rightarrow \rho, \sigma[\quad]}$$

$$\frac{}{(x := \mathbf{new} C, \rho, \sigma) \rightarrow \rho[\quad], \sigma[\quad]}$$

## Structural operational semantics (1/2)

$$\frac{}{(x := \mathbf{null}, \rho, \sigma) \rightarrow \rho[x \mapsto \diamond], \sigma}$$

$$\frac{}{(x := y, \rho, \sigma) \rightarrow \rho[x \mapsto \rho(y)], \sigma}$$

$$\rho(y) \in \mathbf{dom}(\sigma)$$

$$\frac{}{(x := y.f, \rho, \sigma) \rightarrow \rho[x \mapsto \sigma(\rho(y))(f)], \sigma}$$

$$\rho(x) = r \in \mathbf{dom}(\sigma) \quad o' = \sigma(r)[f \mapsto \rho(y)]$$

$$\frac{}{(x.f := y, \rho, \sigma) \rightarrow \rho, \sigma[r \mapsto o']}$$

$$r \notin \mathbf{dom}(\sigma)$$

$$\frac{}{(x := \mathbf{new} C, \rho, \sigma) \rightarrow \rho[x \mapsto r], \sigma[r \mapsto \lambda f. \diamond]}$$

# Structural operational semantics (2/2)

$$(\text{skip}, \rho, \sigma) \rightarrow \rho, \sigma$$

$$(S_1, \rho, \sigma) \rightarrow \rho', \sigma'$$

$$(S_1 ; S_2, \rho, \sigma) \rightarrow (S_2, \rho', \sigma')$$

$$(S_1, \rho, \sigma) \rightarrow (S'_1, \rho', \sigma')$$

$$(S_1 ; S_2, \rho, \sigma) \rightarrow (S'_1 ; S_2, \rho', \sigma')$$

$$(\text{if } (*) \text{ then } S_1 \text{ else } S_2, \rho, \sigma) \rightarrow (S_1, \rho, \sigma)$$

$$(\text{if } (*) \text{ then } S_1 \text{ else } S_2, \rho, \sigma) \rightarrow (S_2, \rho, \sigma)$$

$$(\text{while } (*) \text{ do } S, \rho, \sigma) \rightarrow (S ; \text{while } (*) \text{ do } S, \rho, \sigma)$$

$$(\text{while } (*) \text{ do } S, \rho, \sigma) \rightarrow \rho, \sigma$$

# Vocabulary

Heap expressions :

$$e := x \mid x.f$$

May-alias :

*There exists an execution path to a given program point such that two given memory accesses have the same target.*

Must-alias :

*For all execution paths to a given program point, two given memory accesses have always the same target.*

# Dynamic allocation makes things harder

The set of reachable heaps after a loop like

```
while (*) do ( $x := \mathbf{new\ C}$  ; ...)
```

doesn't have a common bounded codomain.

We need to have a finite representation of something which is infinite...

# Points-to analysis

*Points-to analysis* is a fundamental static analysis to determine the set of objects whose addresses may be stored in variables or fields of objects.

Basic idea : all reference that are created at the same allocation site are merged in a same equivalence class.

We attach an allocation label  $h \in \mathcal{H}$  at each instruction `new C`.

`newh C`

The static object name is then defined as  $\mathcal{O} = \mathcal{H}$ .

# Example

```

class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:   temp.val = new T();
2:   temp.val.f = new A();
3:   temp.next = l;
      l = temp }
    while (*) {
4:   T t = new T();
      t.data = l;
5:   t.start();
      t.f = ...;}
    return;
  }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:   List m = this.data;
7:   while (*) { m = m.next; }
8:   synchronized(m){ m.val.f = ...;}}
    return;}}

```

# Example

```


class List{ T val; List next; }

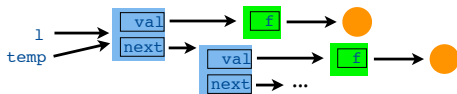
class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
      1: temp.val = new T();
      2: temp.val.f = new A();
      3: temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
      4: t.data = l;
      t.start();
      5: t.f = ...; }
    return;
  }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
      6: List m = this.data;
      7: while (*) { m = m.next; }
      8: synchronized(m){ m.val.f = ...; }
      return; }
}

```

I. We create a link list l

Threads: 



# Example

```

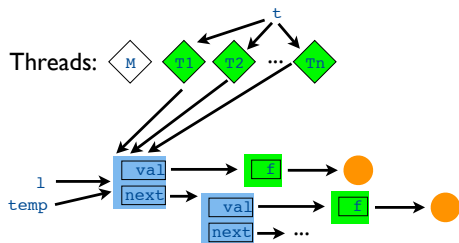
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:   temp.val = new T();
2:   temp.val.f = new A();
3:   temp.next = l;
      l = temp }
4:   while (*) {
      T t = new T();
5:   t.data = l;
      t.start();
6:   t.f = ...;}
    return;
  }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
7:   List m = this.data;
8:   while (*) { m = m.next; }
      synchronized(m){ m.val.f = ...;}}
    return;}}

```

1. We create a link list `l`
2. We create a bunch of thread that all share the list `l`



# Example

```

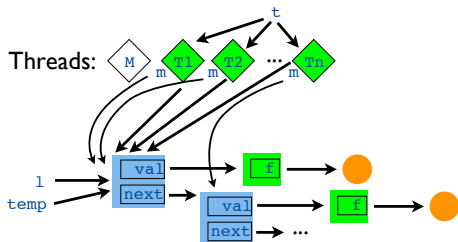
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:   temp.val = new T();
2:   temp.val.f = new A();
3:   temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:   t.data = l;
      t.start();
5:   t.f = ...;
      return;
    }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:   List m = this.data;
7:   while (*) { m = m.next; }
8:   synchronized(m){ m.val.f = ...; }
      return;}}

```

1. We create a link list `l`
2. We create a bunch of thread that all share the list `l`
3. Each thread chooses a cell, takes a lock on it and updates it.



# Example

```

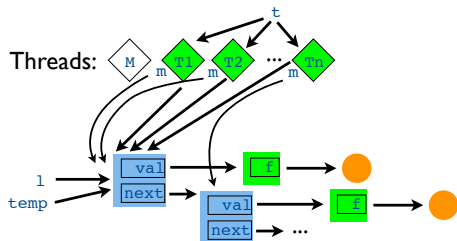
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:   temp.val = new T();
2:   temp.val.f = new A();
3:   temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:   t.data = l;
      t.start();
5:   t.f = ...;}
    return;
  }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:   List m = this.data;
7:   while (*) { m = m.next; }
8:   synchronized(m){ m.val.f = ...;}}
    return;}}

```

1. We create a link list `l`
2. We create a bunch of thread that all share the list `l`
3. Each thread chooses a cell, takes a lock on it and updates it.



# Example

```

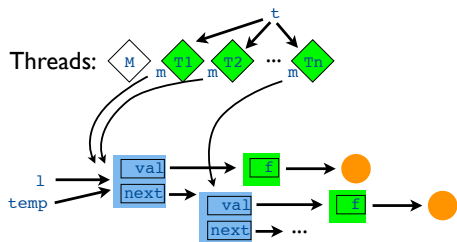
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:   temp.val = new T();
2:   temp.val.f = new A();
3:   temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:   t.data = l;
      t.start();
5:   t.f = ...; }
    return;
  }
}

class T {
  A f;
  List data;
  void run(){
    while(*){
6:   List m = this.data;
7:   while (*) { m = m.next; }
8:   synchronized(m){ m.val.f = ...; }
    return; }
}

```

Points-to analysis computes a finite abstraction of the memory where locations are abstracted by their allocation site



# Example

```

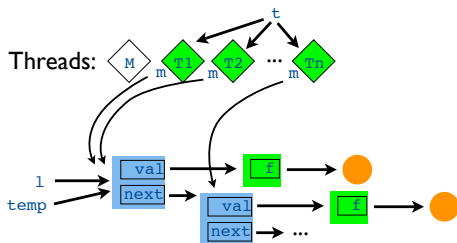
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      [h1] List temp = new List();
1:   [h2] temp.val = new T();
2:   [h3] temp.val.f = new A();
3:   temp.next = l;
      l = temp }
    while (*) {
      [h4] T t = new T();
4:   t.data = l;
      t.start();
5:   t.f = ...;}
    return;
  }
}

class T {
  A f;
  List data;
  void run(){
    while(*){
6:   List m = this.data;
7:   while (*) { m = m.next; }
8:   synchronized(m){ m.val.f = ...;}}
    return;}}

```

Points-to analysis computes a finite abstraction of the memory where locations are abstracted by their allocation site



# Example

```

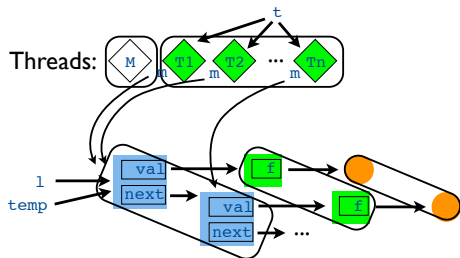
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      [h1] List temp = new List();
1: [h2] temp.val = new T();
2: [h3] temp.val.f = new A();
3:   temp.next = l;
      l = temp }
    while (*) {
      [h4] T t = new T();
4:   t.data = l;
      t.start();
5:   t.f = ...; }
    return;
  }
}

class T {
  A f;
  List data;
  void run(){
    while(*){
6:   List m = this.data;
7:   while (*) { m = m.next; }
8:   synchronized(m){ m.val.f = ...; }
    return; }
}

```

Points-to analysis computes a finite abstraction of the memory where locations are abstracted by their allocation site



# Example

```

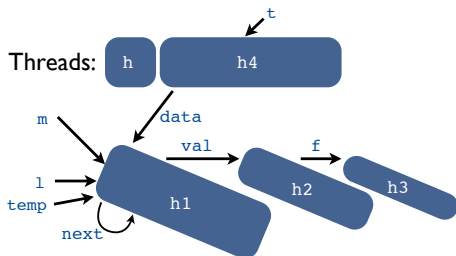
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      [h1] List temp = new List();
1:   [h2] temp.val = new T();
2:   [h3] temp.val.f = new A();
3:   temp.next = l;
      l = temp }
    while (*) {
      [h4] T t = new T();
4:   t.data = l;
      t.start();
5:   t.f = ...;}
    return;
  }
}

class T {
  A f;
  List data;
  void run(){
    while(*){
6:   List m = this.data;
7:   while (*) { m = m.next; }
8:   synchronized(m){ m.val.f = ...;}}
    return;}}

```

Points-to analysis computes a finite abstraction of the memory where locations are abstracted by their allocation site

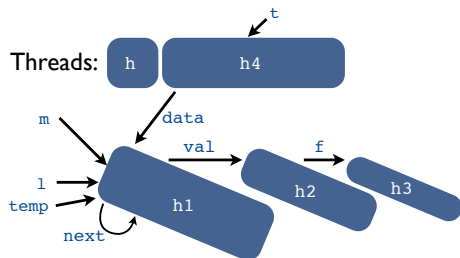
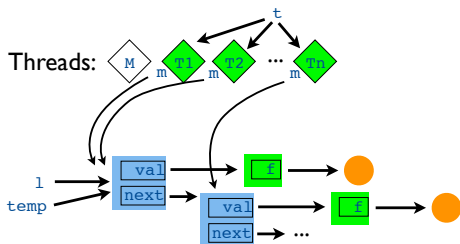


# Example

```
class List{ T val; List next; }
```

```
class Main() {
  void main(){
    List l = null;
    while (*) {
      h1 List temp = new List();
      1: h2 temp.val = new T();
      2: h3 temp.val.f = new A();
      3: temp.next = l;
        l = temp }
      while (*) {
        h4 T t = new T();
      4: t.data = l;
        t.start();
      5: t.f = ...;
      return;
    }
  }
}
```

```
class T {
  A f;
  List data;
  void run(){
    while(*){
      6: List m = this.data;
      7: while (*) { m = m.next; }
      8: synchronized(m){ m.val.f = ...; }
      return; }
}
```



# Points-to analysis

We compute points-to relation for each variable and each fields :

$$(PtV, PtF) \in (Var \rightarrow \wp(\mathcal{O})) \times (Field \rightarrow \wp(\mathcal{O} \times \mathcal{O}))$$

Informal meaning :

- ▶  $h \in PtV(x)$  : the variable  $x$  may contain a reference to an object allocated at site  $h$ .
- ▶  $(h, h') \in PtF(f)$  : the heap may contain an object allocated at site  $h$  whose fields  $f$  points to an object allocated at site  $h'$ .

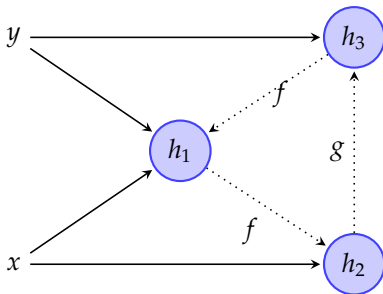
# Points-to graph

Each analysis result  $(PtV, PtF)$  can be represented by a graph.

Example :

$$PtV = [x \mapsto \{h_1, h_2\}, y \mapsto \{h_1, h_3\}]$$

$$PtF = [f \mapsto \{(h_1, h_2), (h_2, h_3)\}, g \mapsto \{(h_3, h_1)\}]$$



# Constraint based specification

Complete lattice structure on  $(\mathbf{Var} \rightarrow \wp(\mathcal{O})) \times (\mathbf{Field} \rightarrow \wp(\mathcal{O} \times \mathcal{O}))$  :

$$\begin{aligned}
 (PtV_1, PtF_1) \sqsubseteq (PtV_2, PtF_2) \text{ iff } & \forall x \in \mathbf{Var}, PtV_1(x) \subseteq PtV_2(x) \\
 & \text{and } \forall f \in \mathbf{Field}, PtF_1(f) \subseteq PtF_2(f)
 \end{aligned}$$

$$\begin{aligned}
 \bigsqcup S = & (\lambda x. \bigcup \{PtV(x) \mid (PtV, PtF) \in S\}, \lambda f. \bigcup \{PtF(f) \mid (PtV, PtF) \in S\}) \\
 & \forall S \subseteq (\mathbf{Var} \rightarrow \wp(\mathcal{O})) \times (\mathbf{Field} \rightarrow \wp(\mathcal{O} \times \mathcal{O}))
 \end{aligned}$$

The analysis of a program  $P$  is then specified as the least solution  $(PtV, PtF)$  of the constraint system

$$PtV, PtF \vdash P$$

# Constraint based specification

$$\frac{}{PtV, PtF \vdash \mathbf{skip}}$$

$$\frac{}{PtV, PtF \vdash x := \mathbf{null}}$$

$$\{h\} \subseteq PtV(x)$$

$$\frac{}{PtV, PtF \vdash x := \mathbf{new}^h C}$$

$$PtV(y) \subseteq PtV(x)$$

$$\frac{}{PtV, PtF \vdash x := y}$$

$$\{h' \mid \exists h \in PtV(y), (h, h') \in PtF(f)\} \subseteq PtV(x)$$

$$\frac{}{PtV, PtF \vdash x := y.f}$$

$$\{(h, h') \mid h \in PtV(x), h' \in PtV(y)\} \subseteq PtF(f)$$

$$\frac{}{PtV, PtF \vdash x.f := y}$$

$$PtV, PtF \vdash S_1 \quad PtV, PtF \vdash S_2$$

$$\frac{}{PtV, PtF \vdash S_1 ; S_2}$$

$$PtV, PtF \vdash S_1 \quad PtV, PtF \vdash S_2$$

$$\frac{}{PtV, PtF \vdash \mathbf{if} (*) \mathbf{then} S_1 \mathbf{else} S_2}$$

$$PtV, PtF \vdash S$$

$$\frac{}{PtV, PtF \vdash \mathbf{while} (*) \mathbf{do} S}$$

## Exercise

Build and solve the constraint system for the points-to analysis of the following program :

```
 $x := \text{new}^{h_1} C ; y := \text{new}^{h_2} C ; \text{if } (*) \text{ then } (x.f := y ; y := x) \text{ else } (y := x.f)$ 
```

# Exercise

Build and solve the constraint system for the points-to analysis of the following program :

```
 $x := \text{new}^{h_1} C ; y := \text{new}^{h_2} C ; \text{if } (*) \text{ then } (x.f := y ; y := x) \text{ else } (y := x.f)$ 
```

$$\{h_1\} \subseteq PtV(x)$$

$$\{h_2\} \subseteq PtV(y)$$

$$\{(h, h') \mid h \in PtV(x), h' \in PtV(y)\} \subseteq PtF(f)$$

$$PtV(x) \subseteq PtV(y)$$

$$\{h' \mid \exists h \in PtV(x), (h, h') \in PtF(f)\} \subseteq PtV(y)$$

## Exercise

Build and solve the constraint system for the points-to analysis of the following program :

```
 $x := \text{new}^{h_1} C ; y := \text{new}^{h_2} C ; \text{if } (*) \text{ then } (x.f := y ; y := x) \text{ else } (y := x.f)$ 
```

$$PtV(x) = \{h_1\}$$

$$PtV(y) = \{h_2\} \cup PtV(x) \cup \{h' \mid \exists h \in PtV(x), (h, h') \in PtF(f)\}$$

$$PtF(f) = \{(h, h') \mid h \in PtV(x), h' \in PtV(y)\}$$

## Exercise

Build and solve the constraint system for the points-to analysis of the following program :

```
 $x := \text{new}^{h_1} C ; y := \text{new}^{h_2} C ; \text{if } (*) \text{ then } (x.f := y ; y := x) \text{ else } (y := x.f)$ 
```

$$PtV(x) = \{h_1\} \quad (1)$$

$$PtV(y) = \{h_1, h_2\} \cup \{h' \mid (h_1, h') \in PtF(f)\} \quad (2)$$

$$PtF(f) = \{(h_1, h') \mid h' \in PtV(y)\} \quad (3)$$

# Exercise

Build and solve the constraint system for the points-to analysis of the following program :

$x := \text{new}^{h_1} C ; y := \text{new}^{h_2} C ; \text{if } (*) \text{ then } (x.f := y ; y := x) \text{ else } (y := x.f)$

$$PtV(x) = \{h_1\} \quad (1)$$

$$PtV(y) = \{h_1, h_2\} \cup \{h' \mid (h_1, h') \in PtF(f)\} \quad (2)$$

$$PtF(f) = \{(h_1, h') \mid h' \in PtV(y)\} \quad (3)$$

		(2)	(3)	
$PtV(y)$	$\emptyset$	$\{h_1, h_2\}$	$\{h_1, h_2\}$	stable
$PtF(f)$	$\emptyset$	$\emptyset$	$\{(h_1, h_1), (h_1, h_2)\}$	stable

## Exercise

Build and solve the constraint system for the points-to analysis of the following program :

$x := \text{new}^{h_1} C ; y := \text{new}^{h_2} C ; \text{if } (*) \text{ then } (x.f := y ; y := x) \text{ else } (y := x.f)$

$$PtV(x) = \{h_1\} \quad (1)$$

$$PtV(y) = \{h_1, h_2\} \cup \{h' \mid (h_1, h') \in PtF(f)\} \quad (2)$$

$$PtF(f) = \{(h_1, h') \mid h' \in PtV(y)\} \quad (3)$$

		(2)	(3)	
$PtV(y)$	$\emptyset$	$\{h_1, h_2\}$	$\{h_1, h_2\}$	stable
$PtF(f)$	$\emptyset$	$\emptyset$	$\{(h_1, h_1), (h_1, h_2)\}$	stable

$$PtV(x) = \{h_1\} \quad PtV(y) = \{h_1, h_2\} \quad PtF(f) = \{(h_1, h_1), (h_1, h_2)\}$$

# Vocabulary

An analysis is said *flow-insensitive* if the order of statements in a program does not affect the result of the analysis.

Example :

- ▶ the points-to analysis seen before is flow-insensitive,
- ▶ the reachable definition analysis is flow-sensitive

## Exercise

Propose a flow-sensitive version (only for local variables) of the previous points-to analysis :

$$(PtV, PtF) \in (\mathbf{Lab} \rightarrow \mathbf{Var} \rightarrow \wp(\mathcal{O})) \times (\mathbf{Field} \rightarrow \wp(\mathcal{O} \times \mathcal{O}))$$

Using the following convenient notation : for  $V, V' \in \mathbf{Var} \rightarrow \wp(\mathcal{O})$ , we define  $V \sqsubseteq_S V'$  by

$$V \sqsubseteq_S V' \text{ iff } \forall y \in S, V(y) \subseteq V'(y)$$

## Exercise

Propose a flow-sensitive version (only for local variables) of the previous points-to analysis :

$$(PtV, PtF) \in (\text{Lab} \rightarrow \text{Var} \rightarrow \wp(\Theta)) \times (\text{Field} \rightarrow \wp(\Theta \times \Theta))$$

Using the following convenient notation : for  $V, V' \in \text{Var} \rightarrow \wp(\Theta)$ , we define  $V \sqsubseteq_s V'$  by

$$V \sqsubseteq_s V' \text{ iff } \forall y \in S, V(y) \subseteq V'(y)$$

$$\frac{PtV(l) \sqsubseteq_{\text{Var}} PtV(l')}{PtV, PtF \vdash [\text{skip}]^l, l'} \quad \frac{PtV(l) \sqsubseteq_{\text{Var} \setminus \{x\}} PtV(l')}{PtV, PtF \vdash [x := \text{null}]^l, l'}$$

$$\frac{\{h\} \subseteq PtV(l')(x) \quad PtV(l) \sqsubseteq_{\text{Var} \setminus \{x\}} PtV(l')}{PtV, PtF \vdash [x := \text{new}^h C]^l, l'}$$

$$\frac{PtV(l)(y) \subseteq PtV(l')(x) \quad PtV(l) \sqsubseteq_{\text{Var} \setminus \{x\}} PtV(l')}{PtV, PtF \vdash [x := y]^l, l'}$$

# Exercise

Propose a flow-sensitive version (only for local variables) of the previous points-to analysis :

$$\frac{\{h' \mid \exists h \in PtV(l)(y), (h, h') \in PtF(f)\} \subseteq PtV(l')(x) \quad PtV(l) \sqsubseteq_{Var \setminus \{x\}} PtV(l')}{PtV, PtF \vdash [x := y.f]^l, l'}$$

$$\frac{\{(h, h') \mid h \in PtV(l)(x), h' \in PtV(l)(y)\} \subseteq PtF(f) \quad PtV(l) \sqsubseteq_{Var} PtV(l')}{PtV, PtF \vdash [x.f := y]^l, l'}$$

$$\frac{PtV, PtF \vdash S_1, \mathit{init}(S_2) \quad PtV, PtF \vdash S_2, l'}{PtV, PtF \vdash S_1 ; S_2, l'}$$

$$\frac{PtV, PtF \vdash S_1, l' \quad PtV(l) \sqsubseteq_{Var} PtV(\mathit{init}(S_1)) \quad PtV, PtF \vdash S_2, l' \quad PtV(l) \sqsubseteq_{Var} PtV(\mathit{init}(S_2))}{PtV, PtF \vdash \mathbf{if} [(*)]^l \mathbf{then} S_1 \mathbf{else} S_2, l'}$$

$$\frac{PtV, PtF \vdash S, l \quad PtV(l) \sqsubseteq_{Var} PtV(\mathit{init}(S)) \quad PtV(l) \sqsubseteq_{Var} PtV(l')}{PtV, PtF \vdash \mathbf{while} [(*)]^l \mathbf{do} S, l'}$$

# Outline

- 1 Constraint based analysis
- 2 Static analysis for Java
- 3 Soundness proof of the points-to analysis

# Soundness proof of the points-to analysis

The soundness of the analysis is proved w.r.t. to an instrumented version of the operational semantics.

We tag references with their allocation sites :

$$\frac{r_{h'} \notin \text{dom}(\sigma) \quad \forall h' \in \mathcal{H}}{(x := \text{new}^h C, \rho, \sigma) \rightarrow \rho[x \mapsto r_h], \sigma[r_h \mapsto \lambda f. \diamond]}$$

This is a safe instrumentation since we obtain exactly the previous semantics by discarding the reference tags.

# Approximation relation

Approximation relation  $\sim$  between a state  $s \in \mathbf{State}$  and an analysis result  $(PtV, PtF)$

$$\begin{aligned}
 (\rho, \sigma) \sim (PtV, PtF) \text{ iff } & \forall x \in \mathbf{Var}, \rho(x) = r_h \in \mathbf{dom}(\sigma) \Rightarrow h \in PtV(x) \\
 \text{and } & \forall r_h \in \mathbf{dom}(\sigma), \forall f \in \mathbf{Field}, \\
 & \sigma(r_h)(f) = r_{h'} \Rightarrow (h, h') \in PtF(f)
 \end{aligned}$$

# Soundness proof

## Lemma (Subject Reduction)

$(s, S) \rightarrow (S', s')$  implies

- ▶ if  $PtV, PtF \vdash S$  and  $s \sim (PtV, PtF)$ ,
- ▶ then  $s' \sim (PtV, PtF)$  and  $PtV, PtF \vdash S'$ .

$(s, S) \rightarrow s'$  implies

- ▶ if  $s \sim (PtV, PtF)$ ,
- ▶ then  $s' \sim (PtV, PtF)$ .

Proof : by induction on  $\rightarrow$ .

# Proof of lemma : a few cases

Assignment :

$$\frac{}{(x := y, \rho, \sigma) \rightarrow \rho[x \mapsto \rho(y)], \sigma}$$

Assume  $PtV(y) \subseteq PtV(x)$ .

Assume  $(\rho, \sigma) \sim (PtV, PtF)$ .

Show  $\rho[x \mapsto \rho(y)], \sigma \sim (PtV, PtF)$ .

# Proof of lemma : a few cases

Sequence (1/2) :

$$\frac{(S_1, \rho, \sigma) \rightarrow \rho', \sigma'}{(S_1 ; S_2, \rho, \sigma) \rightarrow (S_2, \rho', \sigma')}$$

Assume  $PtV, PtF \vdash S_1$  and  $PtV, PtF \vdash S_2$ .

Assume  $(\rho, \sigma) \sim (PtV, PtF)$ .

Assume (IH) that

- ▶ if  $PtV, PtF \vdash S_1$  and  $(\rho, \sigma) \sim (PtV, PtF)$ ,
- ▶ then  $(\rho', \sigma') \sim (PtV, PtF)$ .  
 Show  $\rho', \sigma' \sim (PtV, PtF)$  and  $PtV, PtF \vdash S_2$ .

## Proof of lemma : a few cases

Sequence (2/2) :

$$\frac{(S_1, \rho, \sigma) \rightarrow (S'_1, \rho', \sigma')}{(S_1 ; S_2, \rho, \sigma) \rightarrow (S'_1 ; S_2, \rho', \sigma')}$$

Assume  $PtV, PtF \vdash S_1$  and  $PtV, PtF \vdash S_2$ .

Assume  $(\rho, \sigma) \sim (PtV, PtF)$ .

Assume (IH) that

- ▶ if  $PtV, PtF \vdash S_1$   $(\rho, \sigma) \sim (PtV, PtF)$ ,
- ▶ then  $(\rho', \sigma') \sim (PtV, PtF)$  and  $PtV, PtF \vdash S'_1$ .

Show  $\rho', \sigma' \sim (PtV, PtF)$  and  $PtV, PtF \vdash S'_1 ; S_2$ .

# Soundness proof

Initial state :  $s_0 = (\lambda x. \diamond, \lambda r. \text{undefined})$

## Theorem (Points-to soundness)

If  $PtV, PtF \vdash P$  then

- ▶  $(s_0, P) \rightarrow^* (S', s')$  implies  $s' \sim (PtV, PtF)$
- ▶  $(s_0, P) \rightarrow^* s'$  implies  $s' \sim (PtV, PtF)$

Proof : by induction on  $\rightarrow^*$ .

## Theorem (May-alias soundness)

If  $(\rho, \sigma) \sim (PtV, PtF)$  then for all variable  $x, y \in \text{Var}$ ,  $PtV(x) \cap PtV(y) = \emptyset$  implies  $\rho(x) \neq \rho(y)$ .

# Reading

Sections 1 and 2 of Ana Milanova, Atanas Rountev, and Barbara G. Ryder, *Parameterized Object Sensitivity for Points-to Analysis for Java*, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 14, no. 1, pp. 1-41, January 2005.