# CS 497
# Program Analysis

Ondřej Lhoták

November 21 and 26, 2007

# Program Analysis

Prove properties about runtime behaviour of a program without running it

# Program Analysis

Prove properties about runtime behaviour of a program without running it

## Example properties

- array index bounds
- cast safety
- side-effects
- security violations
- resource leaks

## Applications

- optimizing compilers
- program understanding tools
- refactoring tools
- verification and testing tools

# Outline

**Today:**

Some fundamentals of program analysis

**Next week:**

Analyzing tracematches:
An application of program analysis to program understanding/verification

# Sample "Optimizations"

Constant propagation and folding

```
a = 1;
b = 2;
c = a + b;
```

```
c = 3;
```

Common subexpression elimination

```
a = b + c;
d = b + c;
```

```
a = b + c;
d = a;
```

Unreachable code elimination

```
if(DEBUG)
System.out.println("");
```

```
```

# Sample "Optimizations"

Loop-invariant code motion

```
for(i = 0; i < a.length - foo; i++) {
    sum += a[i];
}
```

```
l = a.length - foo;
for(i = 0; i < l; i++) {
    sum += a[i];
}
```

# Sample "Optimizations"

Check elimination

```
for(i = 0; i < 10; i++) {
    if(a == null) throw new Exception();
    if(i<0 || i>=a.length) throw new Exception();
    a[i] = i;
}
```

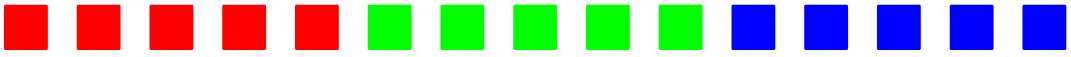
```
for(i = 0; i < 10; i++) {
    a[i] = i;
}
```

# Data Locality Transformations

```
typedef struct {
    int a;
    int b;
    int c;
} foo;
```

Normal layout:

Rearranged layout:



# Data Locality Transformations

```
typedef struct {
    int a;
    int b;
    int c;
} foo;
```

Normal layout:

Rearranged layout:

What if the code contains:

```
foo* f; bar* b = (bar*) f;
```

# A question to ponder...

Q1: What is the output of this program?

```
System.out.println("Hello, World!");
```

# A question to ponder...

Q1: What is the output of this program?

```
System.out.println("Hello, World!");
```

Q2: Given an arbitrary program $p$, can you tell whether its output is "Hello, World!"?

# Does this program print “Hello, World!”?

```
if( arbitraryComputation() ) {
    System.out.println("Hello, World!");
} else {
    System.out.println("Goodbye");
}
```

# Does this program cause an array overflow?

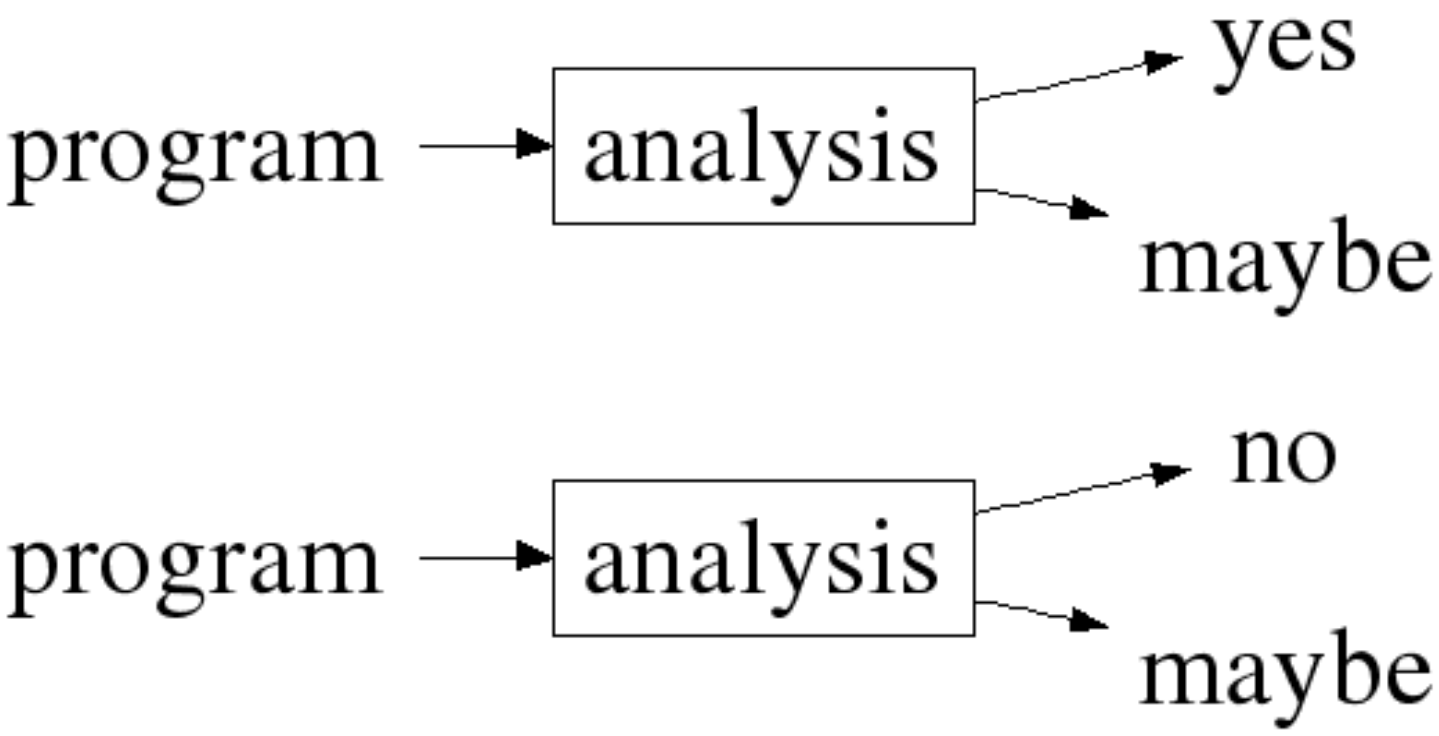
```
if( arbitraryComputation() ) {
    int a[] = new int[5];
    a[10] = 10;
}
```

# Rice's Theorem

For any interesting property $Pr$ of the behaviour of a program, it is impossible to write an analysis that can decide for every program $p$ whether $Pr$ holds for $p$.

# Static Analysis

We settle for static analyses that approximate a property $Pr$.
Example: Does program $p$ access an array out of bounds?




It's always safe to say "maybe"!

# Abstraction Example

```
boolean b = mystery();

if(b) {
    x = 1;
    y = 3;
} else {
    x = 3;
    y = 4;
}

z = x + y;
```

# Abstraction Example

```
boolean b = mystery();
< b is true or false; >
if(b) {
    x = 1;
    y = 3;
} else {
    x = 3;
    y = 4;
}

z = x + y;
```

# Abstraction Example

```
boolean b = mystery();
< b is true or false; >
if(b) {
    x = 1;
    y = 3;
} else {
    x = 3;
    y = 4;
}
< x is 1 or 3; y is 3 or 4; >
z = x + y;
```

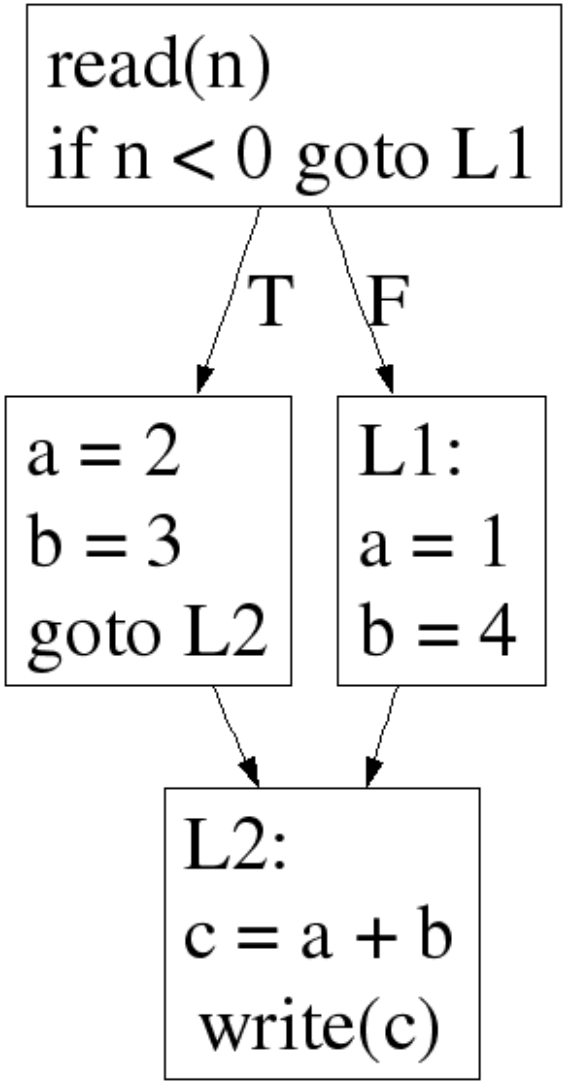
# Abstraction Example

```
boolean b = mystery();
< b is true or false; >
if(b) {
    x = 1;
    y = 3;
} else {
    x = 3;
    y = 4;
}
< x is 1 or 3; y is 3 or 4; >
z = x + y;
< z is 4 or 5 or 6 or 7; >
```

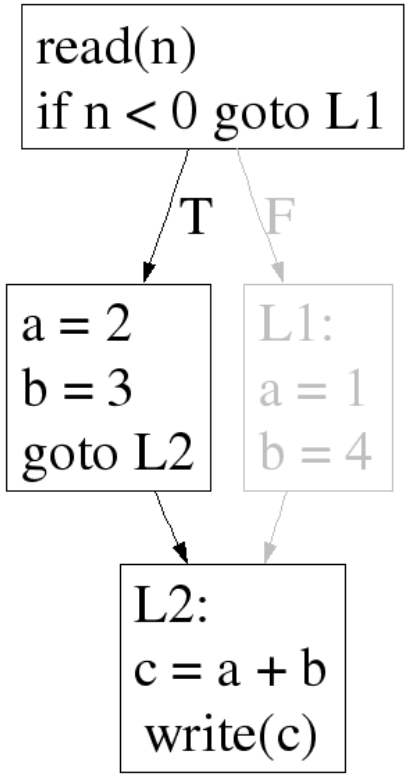
# Code Example

```
     read(n)
     if n < 0 goto L1
     a = 2
     b = 3
     goto L2
L1:  a = 1
     b = 4
L2:  c = a + b
     write(c)
```

# Control Flow Graph

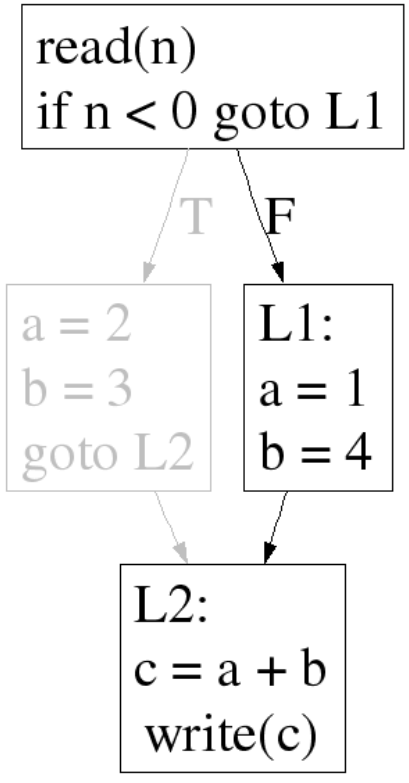


# A Path



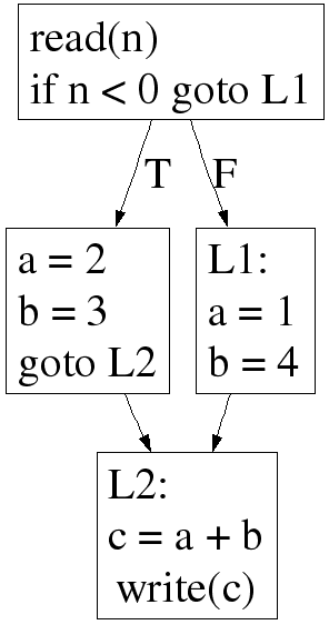

$$f_{\mathsf{write(c)}}(f_{\mathsf{c}=\mathsf{a}+\mathsf{b}}(f_{\mathsf{b}=3}(f_{\mathsf{a}=2}(f_{\mathsf{n}<0}(f_{\mathsf{read(n)}}(\mathsf{init}))))))$$

# Another Path




$$f_{\mathsf{write(c)}}(f_{\mathsf{c}=\mathsf{a}+\mathsf{b}}(f_{\mathsf{b}=4}(f_{\mathsf{a}=1}(f_{\mathsf{n}<0}(f_{\mathsf{read(n)}}(\mathsf{init}))))))$$

# Summarizing Paths




$$f_{\mathsf{write(c)}}(f_{\mathsf{c}=\mathsf{a}+\mathsf{b}}(f_{\mathsf{b}=3}(f_{\mathsf{a}=2}(f_{\mathsf{n}<0}(f_{\mathsf{read(n)}}(\mathsf{init}))))))$$

$$\sqcup$$

$$f_{\mathsf{write(c)}}(f_{\mathsf{c}=\mathsf{a}+\mathsf{b}}(f_{\mathsf{b}=4}(f_{\mathsf{a}=1}(f_{\mathsf{n}<0}(f_{\mathsf{read(n)}}(\mathsf{init}))))))$$

# Definitions

## Definition

A partially ordered set (poset) is a set with a binary relation $\sqsubseteq$ that is

- reflexive ($x \sqsubseteq x$),
- transitive ($x \sqsubseteq y \wedge y \sqsubseteq z \implies x \sqsubseteq z$), and
- antisymmetric ($x \sqsubseteq y \wedge y \sqsubseteq x \implies y = x$).

# Definitions

**Definition**

$z$ is an upper bound of $x$ and $y$ if $x \sqsubseteq z$ and $y \sqsubseteq z$.

**Definition**

$z$ is a least upper bound of $x$ and $y$ if
$z$ is an upper bound of $x$ and $y$, and
for all upper bounds $v$ of $x$ and $y$, $z \sqsubseteq v$.

**Definition**

A lattice is a poset such that for every pair of elements $x, y$, there exists

- a least upper bound $=$ join $= x \sqcup y$, and
- a greatest lower bound $=$ meet $= x \sqcap y$.

# Definitions

**Definition**

In a complete lattice, $\sqcup$ and $\sqcap$ exist for all (possibly infinite) subsets of elements.

**Definition**

A bounded lattice contains two elements:

- $\top$ = top such that $\forall x. x \sqsubseteq \top$
- $\bot$ = bottom such that $\forall x. \bot \sqsubseteq x$

Note: all complete lattices are bounded. (Why?)
Note: all finite lattices are complete. (Why?)

# Definitions

## Powerset Lattice

IF $F$ is a set,
THEN the powerset $\mathcal{P}(F)$ with $\sqsubseteq$ defined as $\subseteq$ (or as $\supseteq$) is a lattice.

# Definitions

## Powerset Lattice

IF $F$ is a set,
THEN the powerset $\mathcal{P}(F)$ with $\sqsubseteq$ defined as $\subseteq$ (or as $\supseteq$) is a lattice.

## Product Lattice

IF $L_A$ and $L_B$ are lattices,
THEN their product $L_A \times L_B$ with $\sqsubseteq$ defined as
$(a_1, b_1) \sqsubseteq (a_2, b_2)$ if $a_1 \sqsubseteq a_2$ and $b_1 \sqsubseteq b_2$ is also a lattice.

# Definitions

## Powerset Lattice

IF $F$ is a set,
THEN the powerset $\mathcal{P}(F)$ with $\sqsubseteq$ defined as $\subseteq$ (or as $\supseteq$) is a lattice.

## Product Lattice

IF $L_A$ and $L_B$ are lattices,
THEN their product $L_A \times L_B$ with $\sqsubseteq$ defined as
$(a_1, b_1) \sqsubseteq (a_2, b_2)$ if $a_1 \sqsubseteq a_2$ and $b_1 \sqsubseteq b_2$ is also a lattice.

## Map Lattice

IF $F$ is a set and $L$ is a lattice,
THEN the map $F \longrightarrow L$ with $\sqsubseteq$ defined as $m_1 \sqsubseteq m_2$ if
$\forall f \in F . m_1(f) \sqsubseteq m_2(f)$ is also a lattice.

# Dataflow Framework

- For each statement $S$ in the control-flow graph, define a $f_S : L \rightarrow L$.

# Dataflow Framework

- For each statement $S$ in the control-flow graph, define a $f_S : L \to L$.
- For a path $P = S_0 S_1 S_2 \ldots S_n$ through the control-flow graph, define $f_P(x) = f_n(\ldots f_2(f_1(f_0(x))))$.

# Dataflow Framework

- For each statement $S$ in the control-flow graph, define a $f_S : L \to L$.
- For a path $P = S_0 S_1 S_2 \ldots S_n$ through the control-flow graph, define $f_P(x) = f_n(\ldots f_2(f_1(f_0(x))))$.
- Goal: find the join-over-all-paths (MOP):

$$\text{MOP}(n, x) = \bigsqcup_{P \text{ is path from } S_0 \text{ to } S_n} f_P(x)$$

# Dataflow Framework

- For each statement $S$ in the control-flow graph, define a $f_S : L \to L$.
- For a path $P = S_0 S_1 S_2 \ldots S_n$ through the control-flow graph, define $f_P(x) = f_n(\ldots f_2(f_1(f_0(x))))$.
- Goal: find the join-over-all-paths (MOP):

$$\text{MOP}(n, x) = \bigsqcup_{P \text{ is path from } S_0 \text{ to } S_n} f_P(x)$$

This is undecidable in general. [Kam, Ullman 1977]

# Dataflow Framework

- For each statement $S$ in the control-flow graph, choose a $f_S : L \rightarrow L$.
- Goal: For each statement $S$ in the control-flow graph, find $V_{Sin} \in L$ and $V_{Sout} \in L$ satisfying:

$$V_{Sout} = f_S(V_{Sin})$$

$$V_{Sin} = \bigsqcup_{P \in PRED(S)} V_{Pout}$$

# Fixed Points

## Fixed Point

$x$ is a fixed point of $F$ if $F(x) = x$.

# Fixed Points

## Fixed Point

$x$ is a fixed point of $F$ if $F(x) = x$.

## Monotone Function

A function $f : L_A \to L_B$ is monotone if
$x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$.

# Fixed Points

## Fixed Point

$x$ is a fixed point of $F$ if $F(x) = x$.

## Monotone Function

A function $f : L_A \rightarrow L_B$ is monotone if
$x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$.

## Knaster-Tarski Fixed Point Theorem

IF $L$ is a complete lattice and $f : L \rightarrow L$ is monotone, THEN the set of fixed points of $f$ is a complete sub-lattice.

$$\bigsqcup_{n \geq 0} f^{(n)}(\bot)$$

is the least fixed point of $L$ (i.e. the $\bot$ of the sub-lattice of fixed points).

# Sketch of Dataflow Algorithm

1. Define a big product lattice

$$\mathcal{L} = \prod_{s \in \text{statements}} L_{s\ in} \times L_{s\ out}$$

2. Define a big function

$$\mathcal{F} : \mathcal{L} \to \mathcal{L}$$

$$\mathcal{F}(V_{s_1 in}, V_{s_1 out}, \ldots) = \left( \bigsqcup_{p \in PRED(s_1)} V_{p\ out}, f_{s_1}(V_{s_1 in}), \ldots \right)$$

3. Iteratively compute least fixed point

$$\bigsqcup_{n \geq 0} \mathcal{F}^{(n)}(\bot)$$

# An Analogy

To solve

$$x = 3x + 4y$$

$$y = 5x + 2y$$

Define

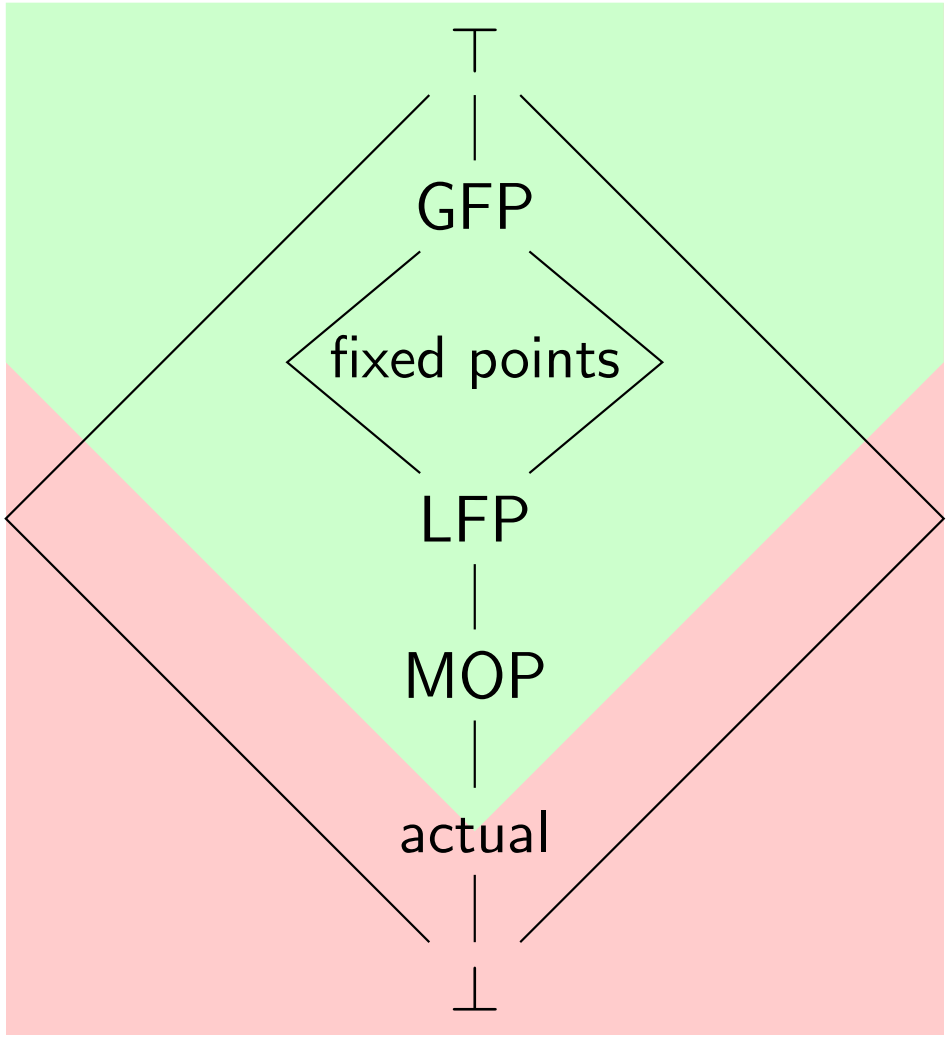
$$F(x, y) = (3x + 4y, 5x + 2y)$$

Find fixed point $(x', y')$ of $F$.
Then

$$(x', y') = F(x', y') = (3x' + 4y', 5x' + 2y')$$

So the fixed point $(x', y')$ solves the system.

# MOP $\sqsubseteq$ LFP




- Every solution $S \sqsupseteq$ actual is safe.
- MOP $\sqsupseteq$ actual
- LFP $\sqsupseteq$ MOP
- Distributive flow function $\Longrightarrow$ LFP $=$ MOP

# MOP vs. LFP

MOP:

$$f_{\mathsf{write(c)}}\big(f_{\mathsf{c}=\mathsf{a}+\mathsf{b}}\big(f_{\mathsf{b}=3}\big(f_{\mathsf{a}=2}\big(f_{\mathsf{n}<0}\big(f_{\mathsf{read(n)}}(\mathsf{init})\big)\big)\big)\big)\big)$$

$$\sqcup$$

$$f_{\mathsf{write(c)}}\big(f_{\mathsf{c}=\mathsf{a}+\mathsf{b}}\big(f_{\mathsf{b}=4}\big(f_{\mathsf{a}=1}\big(f_{\mathsf{n}<0}\big(f_{\mathsf{read(n)}}(\mathsf{init})\big)\big)\big)\big)\big)$$

LFP:

$$f_{\mathsf{write(c)}}\left(f_{\mathsf{c}=\mathsf{a}+\mathsf{b}}\left(\begin{array}{c} f_{\mathsf{b}=3}\big(f_{\mathsf{a}=2}\big(f_{\mathsf{n}<0}\big(f_{\mathsf{read(n)}}(\mathsf{init})\big)\big)\big) \\ \sqcup \\ f_{\mathsf{b}=4}\big(f_{\mathsf{a}=1}\big(f_{\mathsf{n}<0}\big(f_{\mathsf{read(n)}}(\mathsf{init})\big)\big)\big) \end{array}\right)\right)$$

# Distributivity

## Monotone Function

A function $f : L_A \longrightarrow L_B$ is monotone if
$x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$.

## Theorem

IF $f$ is monotone,
THEN $f(x) \sqcup f(y) \sqsubseteq f(x \sqcup y)$.

## Distributive Function

A function $f : L_A \longrightarrow L_B$ is distributive if
$f(x) \sqcup f(y) = f(x \sqcup y)$.

# Designing a Dataflow Analysis

1. Forwards or backwards?
2. What are the lattice elements?
3. Must the property hold on all paths, or must there exist a path?
   (What is the join operator?)
4. On a given path, what are we trying to compute? What are the flow equations?
5. What values hold for program entry points?
6. (What is the initial estimate?)
   It's the unique element $\bot$ such that $\forall x. \bot \sqcup x = x$.

# Interprocedural Analysis Motivation

```
a = 1;
b = 2;


c = a + b;
```

```
a = 1;
b = 2;


c = 3;
```

# Interprocedural Analysis Motivation

```
a = 1;
b = 2;
foo();
c = a + b;
```

```
a = 1;
b = 2;
foo()
c = ???;
```

Does `foo()` modify a or b?

# Interprocedural Analysis Motivation

```
a = 1;
b = 2;
foo();
c = a + b;
```

```
a = 1;
b = 2;
foo()
c = ???;
```

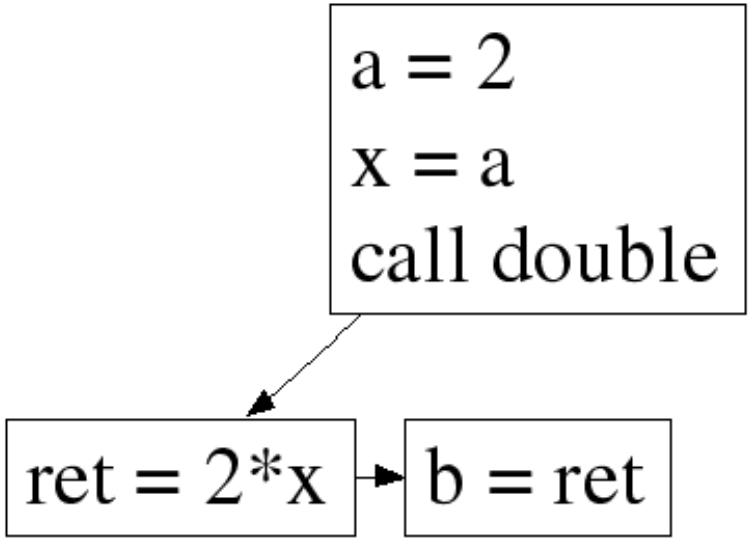
Does `foo()` modify a or b?

```
int double(int x) {
  return 2*x;
}
a = 2;
b = double(a);
c = double(b);
```

Are b and c constant?

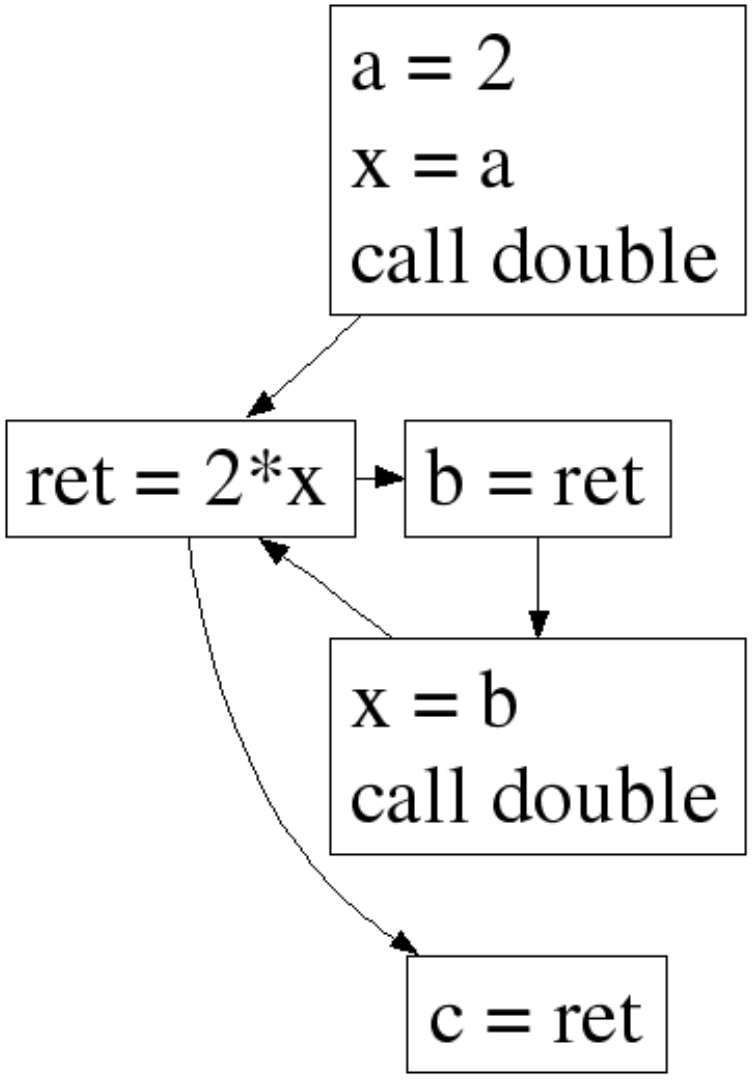
# Control Flow Supergraph

```
a = 2;
b = double(a);
```



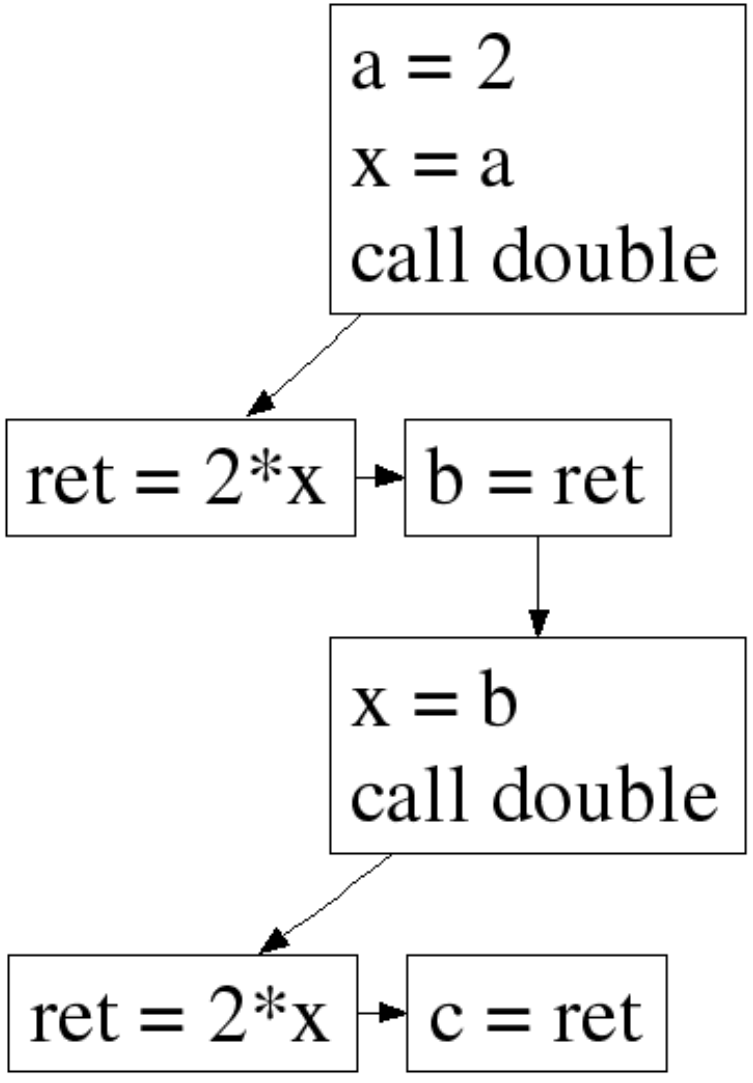
# Control Flow Supergraph

```
a = 2;
b = double(a);
c = double(b);
```



# Context Sensitivity via Cloning

```
a = 2;
b = double(a);
c = double(b);
```



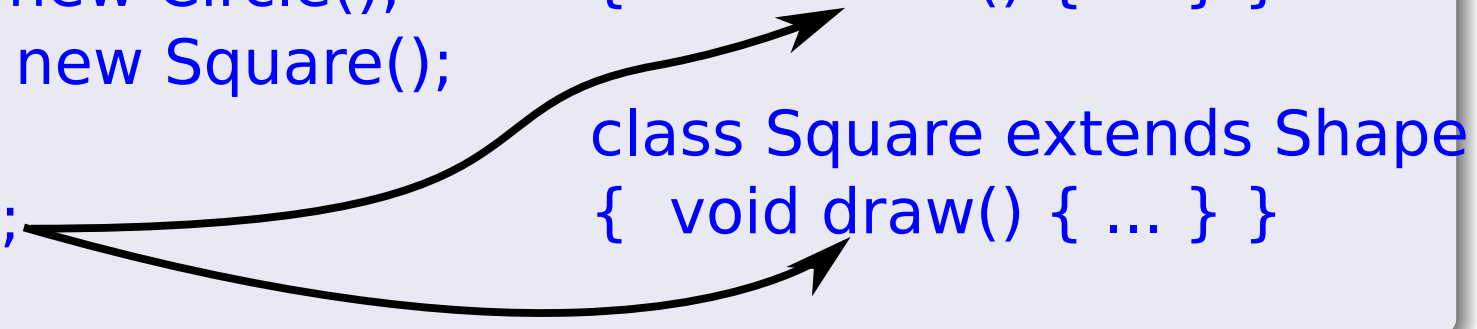
# Call Graph Example

```
Shape s;
if(*) s = new Circle();
else s = new Square();

s.draw();
```

```
class Circle extends Shape
{  void draw() { ... } }

class Square extends Shape
{  void draw() { ... } }
```



## A call graph comprises:

- edges from call sites to methods invoked
- set of reachable methods

# Call Graph Analysis

## Motivation

- (Almost) every interprocedural analysis requires CG
- CG precision affects precision and cost of other analyses

## Open Challenges

- Precision for OO and functional languages
- Analysis of large programs
- Conservative analysis of incomplete programs

# 0CFA: An Analysis for Call Graph Construction

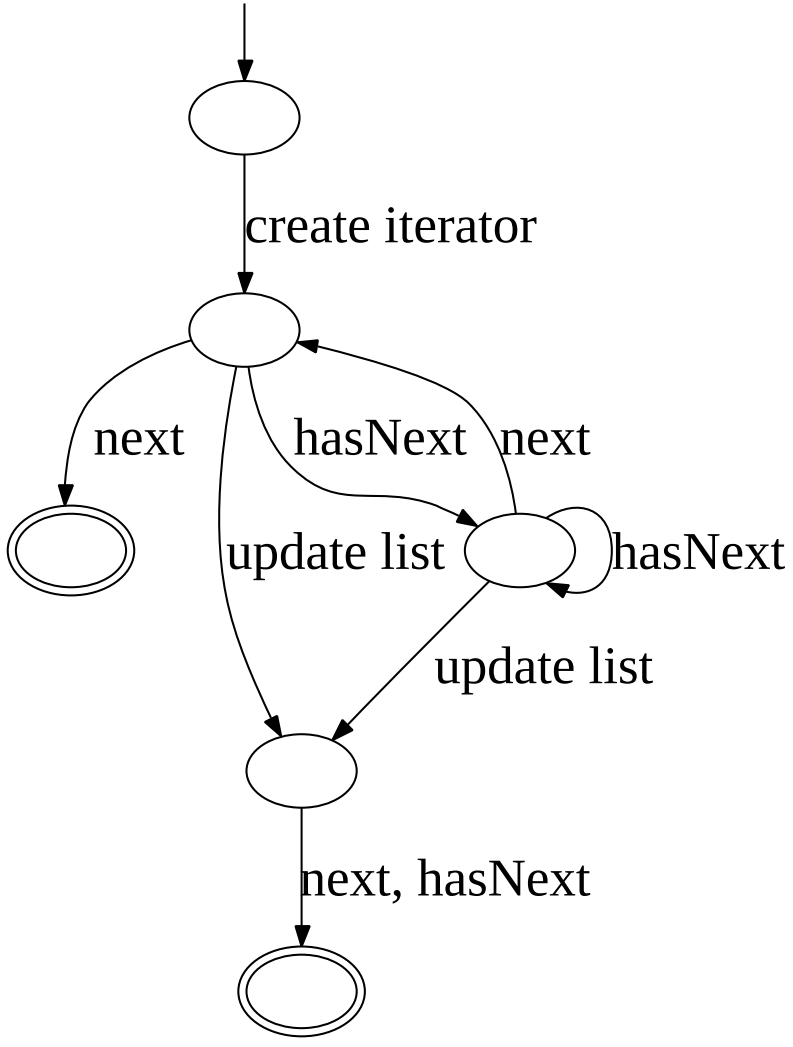
1. Forwards or backwards?
2. What are the lattice elements?
3. Must the property hold on all paths, or must there exist a path?
   (What is the join operator?)
4. On a given path, what are we trying to compute? What are the flow equations?
5. What values hold for program entry points?
6. (What is the initial estimate?)
   It's the unique element $\bot$ such that $\forall x. \bot \sqcup x = x$.

# Tracematch Example

```
List list = new LinkedList();
Iterator it = list.iterator();
while(it.hasNext()) {
    System.out.println(
        it.next());
}
```

# Tracematch Example

```
List list = new LinkedList();
Iterator it = list.iterator();
while(it.hasNext()) {
    System.out.println(
        it.next());
}
```



# Tracematch Analysis

## Motivation

- Document intended usage patterns of interfaces
- Automatically check for violations

## Challenges

- Objects referenced indirectly through pointers
- Tracking multiple interacting objects
- Enough precision to be useful

# File Closing Example

```
tracematch( File f ) {
    sym open after returning(f): call(File open(..));
    sym close after:
            call(void File.close(..)) && target(f);
    sym quit after: execution(void main(String[]));

    open quit

    {
        f.close();
    }
}
```

# Aspect-Oriented Programming

## Purpose

modularize cross-cutting concerns

## How?

An aspect:

- observes program execution for specified events
- executes specified code when events occur

## AspectJ

- aspect-oriented extension of Java

# Definitions

| | |
|---|---|
| joinpoint | An identifiable run-time event (e.g. execution of a method) |
| shadow | Location in program source code of joinpoint start/end |
| pointcut | A predicate (expression) on joinpoints |
| advice | Code to be executed at a joinpoint |
| aspect | A collection of $\langle$pointcut, advice$\rangle$ pairs |

# Tracematches

An aspect comprises:

- advice
- pointcut

A tracematch comprises:

- advice
- an alphabet of symbols, each associated with a pointcut
- a regular language over the symbol alphabet
- a set of parameters

# File Closing Example

```
tracematch( File f ) {
    sym open after returning(f): call(File open(..));
    sym close after:
            call(void File.close(..)) && target(f);
    sym quit after: execution(void main(String[]));

    open quit

    {
        f.close();
    }
}
```

# Declarative semantics: How TMs ought to work

| | trace |
|---|---|
| { | |
| `File a, b, c, d;` | |
| `a = open("X");` | open(X) |
| `d = a;` | |
| `b = open("Y");` | open(Y) |
| `b.close();` | close(Y) |
| `c = open("Z");` | open(Z) |
| `d.close();` | close(X) |
| } | quit |
| | |

Pattern: open quit

# Declarative semantics: How TMs ought to work

| | trace | X | Y | Z |
|---|---|---|---|---|
| `{` | | | | |
| `File a, b, c, d;` | | | | |
| `a = open("X");` | open(X) | open | | |
| `d = a;` | | | | |
| `b = open("Y");` | open(Y) | | open | |
| `b.close();` | close(Y) | | close | |
| `c = open("Z");` | open(Z) | | | open |
| `d.close();` | close(X) | close | | |
| `}` | quit | quit | quit | quit |
| | | no | no | match |

Pattern: open quit

# Subject-Observer Example

```
tracematch(Subject s, Observer o) {
      sym create after returning(o):
         call(Observer.new(..)) && args(s);
      sym update after:
         call(* Subject.update(..)) && target(s);

      create update*

      {
         o.update_view();
      }
}
```

# Operational Semantics 0

## Notation

$\langle Q, A, q_0, Q_f, \delta \rangle$ is the tracematch finite automaton.
$\mathbf{tr}\langle a \quad \rangle$ is a transition statement with symbol $a \in A$

$\mathring{\sigma}$ is a function from states ($Q$) to boolean formulas
. The intended meaning is that the automaton is in state $q$ if and only if the formula $\mathring{\sigma}(q)$ evaluates to true
.

$$\mathring{e} \qquad = \text{true}$$

$$\langle \mathbf{tr}\langle a \quad \rangle, \mathring{\sigma} \rangle \xrightarrow{\circ} \lambda i. \left( \bigvee_{j \,:\, \delta(j,a,i)} \mathring{\sigma}(j) \wedge \mathring{e} \qquad \right) \vee (\mathring{\sigma}(i) \wedge \neg \mathring{e} \qquad )$$

# Operational Semantics 1

## Notation

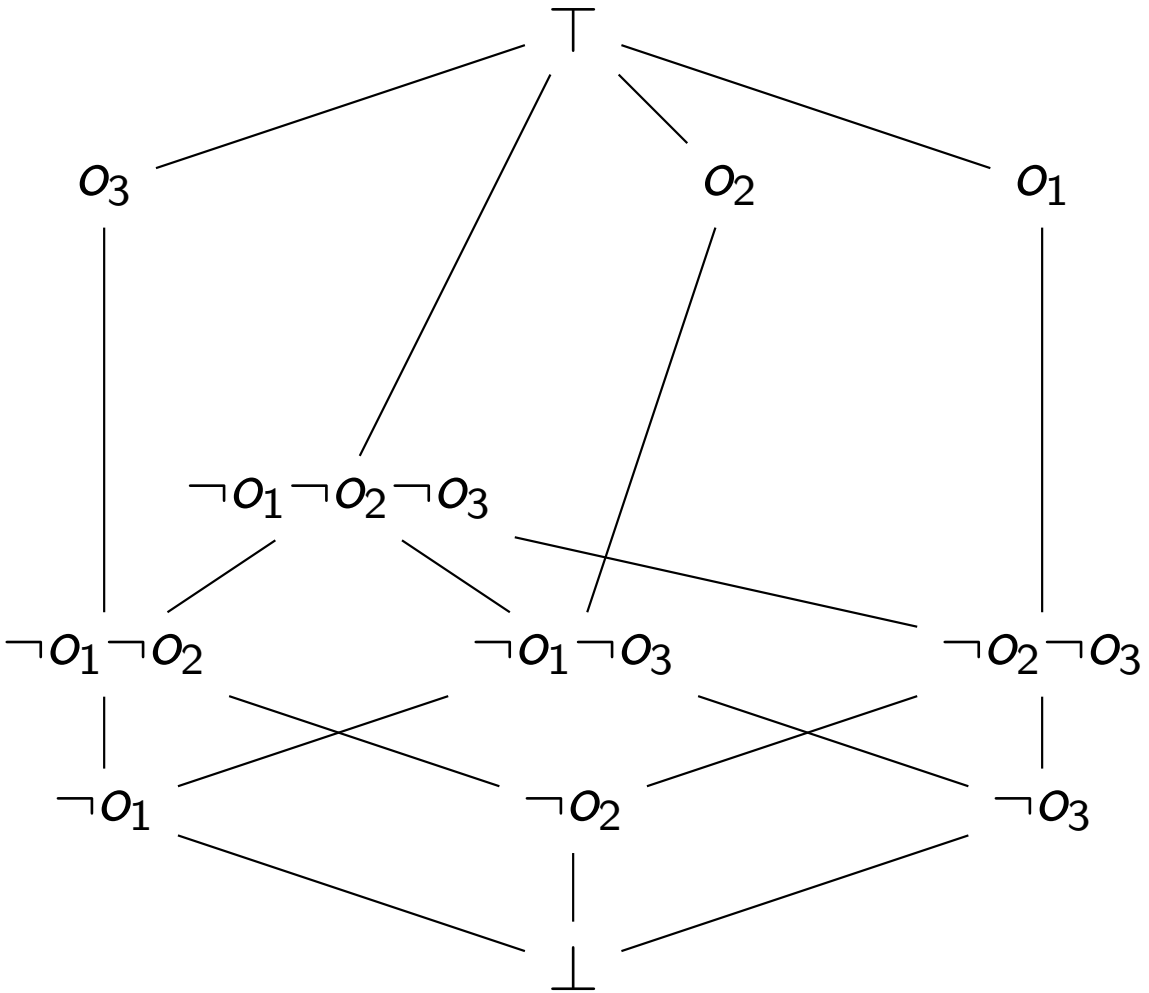
$\langle Q, A, q_0, Q_f, \delta \rangle$ is the tracematch finite automaton.
$\mathbf{tr}\langle a, b \rangle$ is a transition statement with symbol $a \in A$ and variable mapping $b$.
$\mathring{\sigma}$ is a function from states ($Q$) to boolean formulas over parameter values. The intended meaning is that the automaton associated with a given set of parameter values is in state $q$ if and only if the formula $\mathring{\sigma}(q)$ evaluates to true for those parameter values.

$$\mathring{e}(b, \rho) = \bigwedge_{f \in \mathrm{dom}(b)} (f = \rho(b(f)))$$

$$\langle \mathbf{tr} \langle a, b \rangle, \mathring{\sigma} \rangle \xrightarrow{\circ} \lambda i. \left( \bigvee_{j \ : \ \delta(j,a,i)} \mathring{\sigma}(j) \wedge \mathring{e}(b, \rho) \right) \vee (\mathring{\sigma}(i) \wedge \neg \mathring{e}(b, \rho))$$

# Operational Semantics 2

$\sigma \subseteq Q \times (F \rightarrow D)$ where
$Q$ is set of tracematch states,
$F$ is set of tracematch parameters,
$D$ is




positive bindings

negative bindings

# Operational Semantics 2

$$pos(b,\rho)(f) = \begin{cases} \rho(b(f)) & \text{if } f \in \mathrm{dom}(b) \\ \bot & \text{otherwise} \end{cases}$$

$$neg(b,\rho,f)(f') = \begin{cases} \overline{\{\rho(b(f))\}} & \text{if } f = f' \\ \bot & \text{otherwise} \end{cases}$$

$$\langle \mathbf{tr}\,\langle a,b\rangle, \sigma\rangle \rightarrow$$
$$\{\langle i, m \sqcup pos(b,\rho)\rangle \;:\; \langle j,m\rangle \in \sigma \wedge \delta(j,a,i)\} \cup$$
$$\{\langle i, m \sqcup neg(b,\rho,f)\rangle : \langle i,m\rangle \in \sigma \wedge f \in \mathrm{dom}(b)\}$$

# Theorems

Declarative semantics
$\iff$ Operational semantics 1
$\iff$ Operational semantics 2

# Iterator Example

```
tracematch( List l, Iterator i ) {
    sym create after returning(i):
        call(Iterator List.iterator()) && target(l);
    sym update before:
        call(* List.add(..)) && target(l);
    sym next before:
        call(* Iterator.next()) && target(i);

    create next* update next

    {
        throw new ConcurrentModificationException();
    }
}
```

# Analysis 1: Missing Shadows

## Example Code

```
{
    List l = new ArrayList();
    Iterator i = l.iterator(); // create
    l.add(null);               // update
}
```

Pattern: create next* update next

No next $\Longrightarrow$ no match

# Analysis 2: Flow-insensitive Consistent Shadows

## Example Code

```
{
    List l1 = new ArrayList();
    List l2 = new ArrayList();
    Iterator i1 = l1.iterator(); // create
    l2.add(null);                // update
    i1.next();                   // next
}
```

Pattern: create next* update next

No update on l1; no next on l2 $\Longrightarrow$ no match

# Analysis 3: Flow-sensitive Active Shadows

## Example Code

```
{
    List l = new ArrayList();
    Iterator i = l.iterator(); // create
    i.next();                  // next
    l.add(null);               // update
}
```

Pattern: create next* update next

No next after update $\Longrightarrow$ no match

# What do we need to know?

```
List list;
Iterator iter;

while(condition) {
    list = new ArrayList();
    list.add(null);              // update
    iter = list.iterator();      // create
    iter.next();                 // next
}
```

Pattern: create next* update next

Question: Do the two red lists always point to the same object?

# A bigger example

```
void flat(List<List<Integer>> in, List<Integer> out) {
  for(Iterator it = in.iterator(); it.hasNext();) {
    List<Integer> l = it.next();
    for(Iterator it2 = l.iterator(); it2.hasNext();) {
      Integer i = it2.next();
      out.add(i);
    }
  }
}
```

# What we need to know

1. Which variables point to the same/different objects.
2. For each object, which variables point to it at different times.

# Analysis abstraction

- Each concrete object is represented by the set of variables that point to it.

$$\beta_o(o) = \{v \in V \ : \ \rho(v) = o\}$$

- A concrete environment is represented as the set of all abstract objects:

$$\beta_\rho(\rho) = \{\beta_o(o) \ : \ o \text{ is live}\}$$

- Dataflow analysis lattice is $\langle \mathcal{P}(\mathcal{P}(V)), \subseteq \rangle$.
- If a set $o^\sharp$ is not empty, then it represents at most one concrete object: the object pointed to by the variables in $o^\sharp$.

# Analysis transfer function

$$
\begin{aligned}
[\![v_1 \leftarrow v_2]\!]_{o^\sharp}(o^\sharp) &= \begin{cases} o^\sharp \cup \{v_1\} & \text{if } v_2 \in o^\sharp \\ o^\sharp \setminus \{v_1\} & \text{if } v_2 \notin o^\sharp \end{cases} \\
[\![v_1 \leftarrow v_2]\!]_{\rho^\sharp}(\rho^\sharp) &= \left\{ [\![v_1 \leftarrow v_2]\!]_{o^\sharp}(o^\sharp) \;:\; o^\sharp \in \rho^\sharp \right\} \\
[\![v \leftarrow \textbf{new}]\!]_{o^\sharp}(o^\sharp) &= o^\sharp \setminus \{v\} \\
[\![v \leftarrow \textbf{new}]\!]_{\rho^\sharp}(\rho^\sharp) &= \left\{ [\![v_1 \leftarrow v_2]\!]_{o^\sharp}(o^\sharp) \;:\; o^\sharp \in \rho^\sharp \right\} \cup \{\{v\}\}
\end{aligned}
$$

# Analysis transfer function

$$
\begin{aligned}
[\![v_1 \leftarrow v_2]\!]_{o^\sharp}(o^\sharp) &= \begin{cases} o^\sharp \cup \{v_1\} & \text{if } v_2 \in o^\sharp \\ o^\sharp \setminus \{v_1\} & \text{if } v_2 \notin o^\sharp \end{cases} \\
[\![v_1 \leftarrow v_2]\!]_{\rho^\sharp}(\rho^\sharp) &= \left\{ [\![v_1 \leftarrow v_2]\!]_{o^\sharp}(o^\sharp) \;:\; o^\sharp \in \rho^\sharp \right\} \\
[\![v \leftarrow \textbf{new}]\!]_{o^\sharp}(o^\sharp) &= o^\sharp \setminus \{v\} \\
[\![v \leftarrow \textbf{new}]\!]_{\rho^\sharp}(\rho^\sharp) &= \left\{ [\![v_1 \leftarrow v_2]\!]_{o^\sharp}(o^\sharp) \;:\; o^\sharp \in \rho^\sharp \right\} \cup \{\{v\}\} \\
[\![v \leftarrow \text{heap field}]\!]_{\rho^\sharp}(\rho^\sharp) &= ???
\end{aligned}
$$

# Analysis transfer function

$$
\begin{aligned}
[\![v_1 \leftarrow v_2]\!]_{o^\sharp}(o^\sharp) &= \begin{cases} o^\sharp \cup \{v_1\} & \text{if } v_2 \in o^\sharp \\ o^\sharp \setminus \{v_1\} & \text{if } v_2 \notin o^\sharp \end{cases} \\
[\![v_1 \leftarrow v_2]\!]_{\rho^\sharp}(\rho^\sharp) &= \left\{ [\![v_1 \leftarrow v_2]\!]_{o^\sharp}(o^\sharp) \ : \ o^\sharp \in \rho^\sharp \right\} \\
[\![v \leftarrow \textbf{new}]\!]_{o^\sharp}(o^\sharp) &= o^\sharp \setminus \{v\} \\
[\![v \leftarrow \textbf{new}]\!]_{\rho^\sharp}(\rho^\sharp) &= \left\{ [\![v_1 \leftarrow v_2]\!]_{o^\sharp}(o^\sharp) \ : \ o^\sharp \in \rho^\sharp \right\} \cup \{\{v\}\} \\
[\![v \leftarrow \text{heap field}]\!]_{\rho^\sharp}(\rho^\sharp) &= \rho^\sharp \cup \{o^\sharp \cup \{v\} \ : \ o^\sharp \in \rho^\sharp\}
\end{aligned}
$$

# What the abstraction tells us

## Must aliasing

If $v_1$ and $v_2$ always point to the same object, then every $o^\sharp \in \rho^\sharp$ contains either both $v_1$ and $v_2$, or neither.

## May aliasing

If $v_1$ and $v_2$ cannot point to the same object, then no $o^\sharp \in \rho^\sharp$ contains both $v_1$ and $v_2$.

## Lemma (same object at different times)

If $o^\sharp = \beta_o[\rho](o)$ is the abstraction of some concrete object $o$ before statement $s$, then $o'^\sharp = [\![s]\!]_o(o^\sharp)$ is the abstraction of the same concrete object after statement $s$.

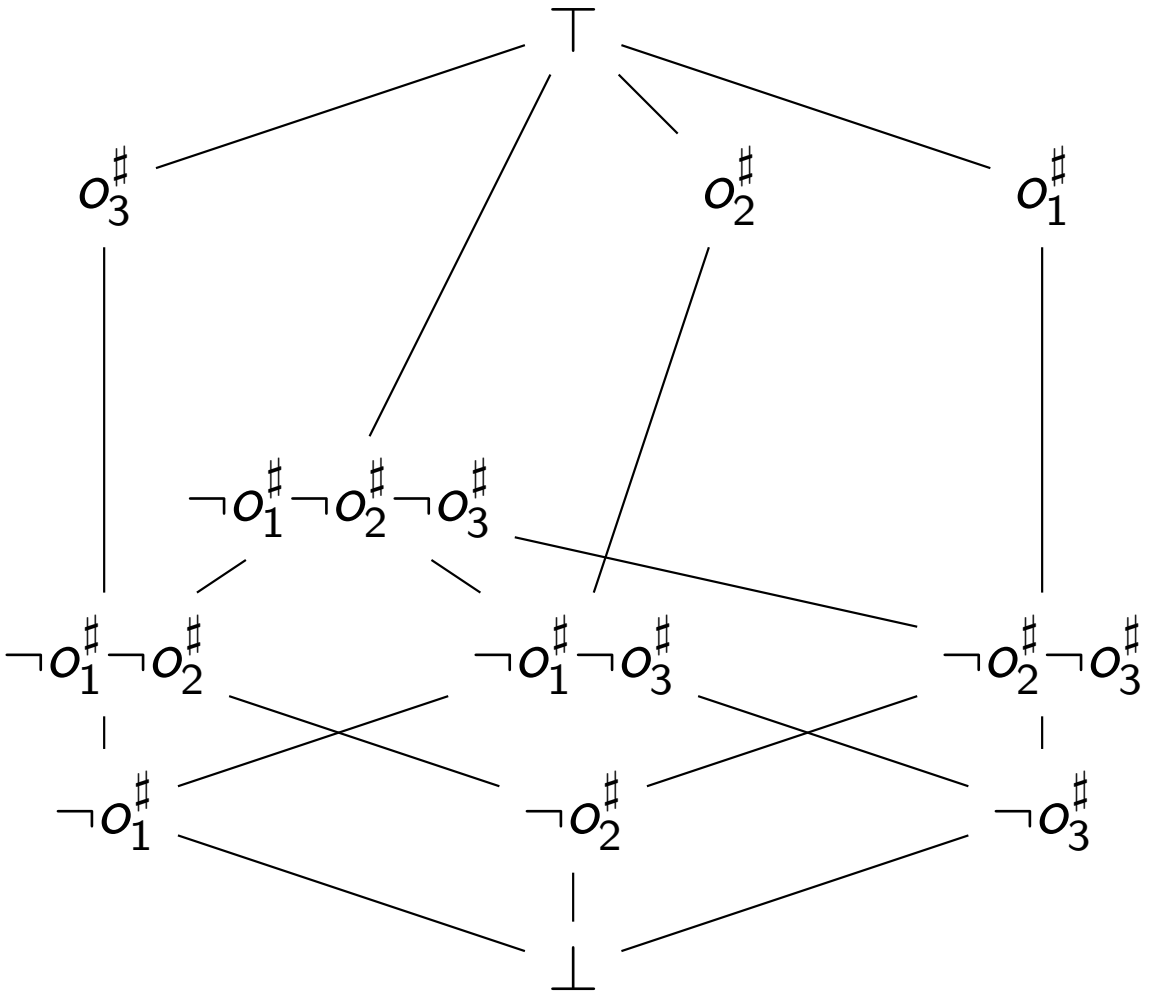
$$\langle s, \rho \rangle \to \rho' \implies [\![s]\!]_{o^\sharp}(\beta_o[\rho](o)) = \beta_o[\rho'](o)$$

# Analysis Soundness Theorem

$$\langle s, \rho \rangle \to \rho' \implies \beta_\rho(\rho') \sqsubseteq [\![s]\!]_{\rho^\sharp}(\beta_\rho(\rho))$$

# Tracematch state abstraction

$\sigma^\sharp \subseteq Q \times (F \to D^\sharp)$ with $\sqsubseteq = \subseteq$ where
$Q$ is set of tracematch states,
$F$ is set of tracematch parameters,
$D^\sharp$ is




positive bindings

negative bindings

# Tracematch state abstraction

$$pos^\sharp(b, \rho^\sharp)(f) = \begin{cases} \rho^\sharp(b(f)) & \text{if } f \in \mathrm{dom}(b) \\ \bot & \text{otherwise} \end{cases}$$

$$neg^\sharp(b, \rho^\sharp, f)(f') = \begin{cases} \overline{\{\rho^\sharp(b(f))\}} & \text{if } f = f' \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\mathbf{tr}\,\langle a, b\rangle]\!]_{\sigma^\sharp}(\sigma^\sharp) =$$

$$\left\{\left\langle i, m^\sharp \sqcup pos^\sharp(b, \rho^\sharp)\right\rangle \;:\; \left\langle j, m^\sharp\right\rangle \in \sigma^\sharp \wedge \delta(j, a, i)\right\} \cup$$
$$\left\{\left\langle i, m^\sharp \sqcup neg(b, \rho^\sharp, f)\right\rangle : \left\langle i, m^\sharp\right\rangle \in \sigma^\sharp \wedge f \in \mathrm{dom}(b)\right\}$$

# Tracematch state abstraction

$$pos^\sharp(b, O^\sharp)(f) = \begin{cases} O^\sharp(b(f)) & \text{if } f \in \mathrm{dom}(b) \\ \bot & \text{otherwise} \end{cases}$$

$$neg^\sharp(b, O^\sharp, f)(f') = \begin{cases} \overline{\{O^\sharp(b(f))\}} & \text{if } f = f' \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\mathbf{tr}\,\langle a, b\rangle]\!]_{\sigma^\sharp}(\sigma^\sharp) = \bigcup_{O^\sharp \subseteq \rho^\sharp \wedge \mathrm{compat}(O^\sharp)} \left\{ \left\langle i, m^\sharp \sqcup pos^\sharp(b, O^\sharp) \right\rangle \;:\; \left\langle j, m^\sharp \right\rangle \in \sigma^\sharp \wedge \delta(j, a, i) \right\} \cup \left\{ \left\langle i, m^\sharp \sqcup neg(b, O^\sharp, f) \right\rangle : \left\langle i, m^\sharp \right\rangle \in \sigma^\sharp \wedge f \in \mathrm{dom}(b) \right\}$$

# Tracematch state abstraction

$$pos^\sharp(b, O^\sharp)(f) = \begin{cases} O^\sharp(b(f)) & \text{if } f \in \mathrm{dom}(b) \\ \bot & \text{otherwise} \end{cases}$$

$$neg^\sharp(b, O^\sharp, f)(f') = \begin{cases} \overline{\{O^\sharp(b(f))\}} & \text{if } f = f' \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\mathbf{tr}\,\langle a, b\rangle]\!]_{\sigma^\sharp}(\sigma^\sharp) = \bigcup_{O^\sharp \subseteq \rho^\sharp \wedge \mathsf{compat}(O^\sharp)}$$

$$\left\{\left\langle i, m^\sharp \sqcup pos^\sharp(b, O^\sharp)\right\rangle \;:\; \left\langle j, m^\sharp\right\rangle \in \sigma^\sharp \wedge \delta(j, a, i)\right\} \cup$$

$$\left\{\left\langle i, m^\sharp \sqcup neg(b, O^\sharp, f)\right\rangle : \left\langle i, m^\sharp\right\rangle \in \sigma^\sharp \wedge f \in \mathrm{dom}(b)\right\}$$

$$[\![s]\!]_{\sigma^\sharp}(\sigma^\sharp) = \mathsf{map}_{o^\sharp}[[\![s]\!]_{o^\sharp}](\sigma^\sharp)$$

# Tracematch state abstraction

$$pos^\sharp(b, O^\sharp)(f) = \begin{cases} O^\sharp(b(f)) & \text{if } f \in \mathrm{dom}(b) \\ \bot & \text{otherwise} \end{cases}$$

$$neg^\sharp(b, O^\sharp, f)(f') = \begin{cases} \overline{\{O^\sharp(b(f))\}} & \text{if } f = f' \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\mathbf{tr}\,\langle a, b\rangle]\!]_{\sigma^\sharp}(\sigma^\sharp) =$$

$$\bigcup_{O^\sharp \subseteq \rho^\sharp \wedge \mathsf{compat}(O^\sharp)}$$

$$\left\{\left\langle i, m^\sharp \sqcup pos^\sharp(b, O^\sharp)\right\rangle \;:\; \left\langle j, m^\sharp\right\rangle \in \sigma^\sharp \wedge \delta(j, a, i)\right\} \cup$$

$$\left\{\left\langle i, m^\sharp \sqcup neg(b, O^\sharp, f)\right\rangle : \left\langle i, m^\sharp\right\rangle \in \sigma^\sharp \wedge f \in \mathrm{dom}(b)\right\}$$

$$[\![s]\!]_{\sigma^\sharp}(\sigma^\sharp) = \mathsf{map}_{o^\sharp}[[\![s]\!]_{o^\sharp}](\sigma^\sharp)$$

What about loads from heap?

# Analysis Soundness Theorem

$$\langle s, \sigma \rangle \to \sigma' \implies \beta_\sigma(\sigma') \sqsubseteq [\![s]\!]_{\sigma^\sharp}(\beta_\sigma(\sigma))$$

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
  for(Iterator it = in.iterator();
                                  it.hasNext();) {
    List<Integer> l = it.next();
    for(Iterator it2 = in.iterator();
                                     it2.hasNext();) {
      Integer i = it2.next();
      out.add(i);
    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
{{in},{out}},{⟨0,⊥,⊥⟩}
  for(Iterator it = in.iterator();
                                    it.hasNext();) {
    List<Integer> l = it.next();
    for(Iterator it2 = in.iterator();
                                       it2.hasNext();) {
      Integer i = it2.next();
      out.add(i);
    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
{{in},{out}},{⟨0,⊥,⊥⟩}
  for(Iterator it = in.iterator();
{{in},{out},{it}              },{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩              }
                                    it.hasNext();) {
{{in},{out},{it}              },{⟨0,⊥,⊥⟩,⟨4,{in},{it}⟩              }
    List<Integer> l = it.next();

    for(Iterator it2 = in.iterator();

                                        it2.hasNext();) {

      Integer i = it2.next();

      out.add(i);

    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
{{in},{out}},{⟨0,⊥,⊥⟩}
  for(Iterator it = in.iterator();
{{in},{out},{it}              },{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩              }
                                    it.hasNext();) {
{{in},{out},{it}              },{⟨0,⊥,⊥⟩,⟨4,{in},{it}⟩              }
    List<Integer> l = it.next();
{{in},{out},{it},{l}          },{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩              }
    for(Iterator it2 = in.iterator();

                                          it2.hasNext();) {

      Integer i = it2.next();

      out.add(i);

    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
{{in},{out}},{⟨0,⊥,⊥⟩}
  for(Iterator it = in.iterator();
{{in},{out},{it}              },{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩              }
                                    it.hasNext();) {
{{in},{out},{it}              },{⟨0,⊥,⊥⟩,⟨4,{in},{it}⟩              }
    List<Integer> l = it.next();
{{in},{out},{it},{l}          },{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩              }
    for(Iterator it2 = in.iterator();
{{in},{out},{it},{l},{it2}    },{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨1,{l},{it2}⟩          }
                                      it2.hasNext();) {

      Integer i = it2.next();

      out.add(i);

    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
```
$\{\{in\},\{out\}\},\{\langle 0,\bot,\bot\rangle\}$
```
  for(Iterator it = in.iterator();
```
$\{\{in\},\{out\},\{it\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{in\},\{it\}\rangle\}$
```
                                  it.hasNext();) {
```
$\{\{in\},\{out\},\{it\}\},\{\langle 0,\bot,\bot\rangle,\langle 4,\{in\},\{it\}\rangle\}$
```
    List<Integer> l = it.next();
```
$\{\{in\},\{out\},\{it\},\{l\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{in\},\{it\}\rangle\}$
```
    for(Iterator it2 = in.iterator();
```
$\{\{in\},\{out\},\{it\},\{l\},\{it2\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{in\},\{it\}\rangle,\langle 1,\{l\},\{it2\}\rangle\}$
```
                                     it2.hasNext();) {
```
$\{\{in\},\{out\},\{it\},\{l\},\{it2\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{in\},\{it\}\rangle,\langle 4,\{l\},\{it2\}\rangle\}$
```
      Integer i = it2.next();

      out.add(i);

    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
```
$\{\{\text{in}\},\{\text{out}\}\},\{\langle 0,\bot,\bot\rangle\}$
```
  for(Iterator it = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle\}$
```
                                 it.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\}\},\{\langle 0,\bot,\bot\rangle,\langle 4,\{\text{in}\},\{\text{it}\}\rangle\}$
```
    List<Integer> l = it.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle\}$
```
    for(Iterator it2 = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
                                    it2.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 4,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
      Integer i = it2.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
      out.add(i);

    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
```
$\{\{\text{in}\},\{\text{out}\}\},\{\langle 0,\bot,\bot\rangle\}$
```
  for(Iterator it = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle\}$
```
                                  it.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\}\},\{\langle 0,\bot,\bot\rangle,\langle 4,\{\text{in}\},\{\text{it}\}\rangle\}$
```
    List<Integer> l = it.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle\}$
```
    for(Iterator it2 = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
                                     it2.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 4,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
      Integer i = it2.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
      out.add(i);
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
```
$\{\{\text{in}\},\{\text{out}\}\},\{\langle 0,\bot,\bot\rangle\}$
```
  for(Iterator it = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
                                    it.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\}\},\{\langle 0,\bot,\bot\rangle,\langle 4,\{\text{in}\},\{\text{it}\}\rangle\}$
```
    List<Integer> l = it.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle\}$
```
    for(Iterator it2 = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
                                       it2.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 4,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
      Integer i = it2.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
      out.add(i);
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
{{in},{out}},{⟨0,⊥,⊥⟩}
  for(Iterator it = in.iterator();
{{in},{out},{it},{l},{it2},{i}},{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨1,{l},{it2}⟩}
                                  it.hasNext();) {
{{in},{out},{it},{l},{it2},{i}},{⟨0,⊥,⊥⟩,⟨4,{in},{it}⟩,⟨1,{l},{it2}⟩}
    List<Integer> l = it.next();
{{in},{out},{it},{l}          },{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩              }
    for(Iterator it2 = in.iterator();
{{in},{out},{it},{l},{it2}    },{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨1,{l},{it2}⟩        }
                                    it2.hasNext();) {
{{in},{out},{it},{l},{it2}    },{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨4,{l},{it2}⟩        }
      Integer i = it2.next();
{{in},{out},{it},{l},{it2},{i}},{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨1,{l},{it2}⟩        }
      out.add(i);
{{in},{out},{it},{l},{it2},{i}},{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨1,{l},{it2}⟩        }
    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
{{in},{out}},{⟨0,⊥,⊥⟩}
  for(Iterator it = in.iterator();
{{in},{out},{it},{l},{it2},{i}},{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨1,{l},{it2}⟩}
                                  it.hasNext();) {
{{in},{out},{it},{l},{it2},{i}},{⟨0,⊥,⊥⟩,⟨4,{in},{it}⟩,⟨1,{l},{it2}⟩}
    List<Integer> l = it.next();
{{in},{out},{it},{l},{it2},{i}},{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨1,{},{it2}⟩}
    for(Iterator it2 = in.iterator();
{{in},{out},{it},{l},{it2}   },{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨1,{l},{it2}⟩          }
                                    it2.hasNext();) {
{{in},{out},{it},{l},{it2}   },{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨4,{l},{it2}⟩          }
      Integer i = it2.next();
{{in},{out},{it},{l},{it2},{i}},{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨1,{l},{it2}⟩          }
      out.add(i);
{{in},{out},{it},{l},{it2},{i}},{⟨0,⊥,⊥⟩,⟨1,{in},{it}⟩,⟨1,{l},{it2}⟩          }
    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
```
$\{\{\text{in}\},\{\text{out}\}\},\{\langle 0,\bot,\bot\rangle\}$
```
  for(Iterator it = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
                                        it.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 4,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
    List<Integer> l = it.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\},\{\text{it2}\}\rangle\}$
```
    for(Iterator it2 = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle,\langle 1,\{\},\{\}\rangle\}$
```
                                          it2.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 4,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
      Integer i = it2.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
      out.add(i);
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
```
$\{\{\text{in}\},\{\text{out}\}\},\{\langle 0,\bot,\bot\rangle\}$
```
  for(Iterator it = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
                                        it.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 4,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
    List<Integer> l = it.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\},\{\text{it2}\}\rangle\}$
```
    for(Iterator it2 = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle,\langle 1,\{\},\{\}\rangle\}$
```
                                          it2.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 4,\{\text{l}\},\{\text{it2}\}\rangle,\langle 1,\{\},\{\}\rangle\}$
```
      Integer i = it2.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
      out.add(i);
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
```
$\{\{\text{in}\},\{\text{out}\}\},\{\langle 0,\bot,\bot\rangle\}$
```
  for(Iterator it = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
                                        it.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 4,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
    List<Integer> l = it.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\},\{\text{it2}\}\rangle\}$
```
    for(Iterator it2 = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle,\langle 1,\{\},\{\}\rangle\}$
```
                                          it2.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 4,\{\text{l}\},\{\text{it2}\}\rangle,\langle 1,\{\},\{\}\rangle\}$
```
      Integer i = it2.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle,\langle 1,\{\},\{\}\rangle\}$
```
      out.add(i);
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
    }
  }
}
```

# List flattening example

```
void flat(List<List<Integer>> in, List<Integer> out) {
```
$\{\{\text{in}\},\{\text{out}\}\},\{\langle 0,\bot,\bot\rangle\}$
```
  for(Iterator it = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
                                        it.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 4,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle\}$
```
    List<Integer> l = it.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\},\{\text{it2}\}\rangle\}$
```
    for(Iterator it2 = in.iterator();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle,\langle 1,\{\},\{\}\rangle\}$
```
                                          it2.hasNext();) {
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 4,\{\text{l}\},\{\text{it2}\}\rangle,\langle 1,\{\},\{\}\rangle\}$
```
      Integer i = it2.next();
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle,\langle 1,\{\},\{\}\rangle\}$
```
      out.add(i);
```
$\{\{\text{in}\},\{\text{out}\},\{\text{it}\},\{\text{l}\},\{\text{it2}\},\{\text{i}\}\},\{\langle 0,\bot,\bot\rangle,\langle 1,\{\text{in}\},\{\text{it}\}\rangle,\langle 1,\{\text{l}\},\{\text{it2}\}\rangle,\langle 1,\{\},\{\}\rangle\}$
```
    }
  }
}
```

# Other Issues

- Interprocedural Analysis
- Implementation

# Experiments: Tracematches Analyzed

| | |
|---|---|
| ASyncIteration | only iterate synchronized collection with lock |
| FailSafeEnum | do not update a vector while iterating over it |
| FailSafeIter | do not update a collection while iterating over it |
| HashMap | do not change an hash code while object in hash map |
| HasNextElem | call hasNextElem before nextElement on Enumeration |
| HasNext | call hasNext before next on Iterator |
| LeakingSync | only access a synchronized collection using wrapper |
| Reader | don't use Reader after InputStream was closed |
| Writer | don't use Writer after OutputStream was closed |

# Experiments: Benchmarks Analyzed

from dacapobench.org

| | |
|---|---|
| antlr | parser generator |
| bloat | Java bytecode optimizer |
| chart | chart plotter with PDF output |
| fop | XSL-FO to PDF transformer |
| hsqldb | in-memory database |
| jython | Python interpreter |
| luindex | document indexer and search |
| pmd | Java source code analyzer |
| xalan | XML to HTML transformer |

# Results

- don't fully know yet...
- very precise
- scalability still needs some work
- some implementation bugs remaining