universitätbonn ROOTS

Context Sensitive Points-to Analysis

Saad Bashir Alvi

alvi@cs.uni-bonn.de

Context-Sensitive Formulation

- A language L_c that filters unrealizable paths.
- It only needs to model program semantics related to method calls and returns.
- A field-sensitive and context-sensitive points-to analysis can be formulated as reachability over the intersection of L_F and L_c.
- Formulation here both filters unrealizable paths and yields a context- sensitive heap abstraction.

Context-Sensitive Formulation

- A language L_c that filters unrealizable paths.
- It only needs to model program semantics related to method calls and returns.
- A field-sensitive and context-sensitive points-to analysis can be formulated as reachability over the intersection of L_F and L_c.
- Formulation here both filters unrealizable paths and yields a context- sensitive heap abstraction.
- Specifying L_c

$$\begin{aligned} callEntry[i] &\rightarrow \mathsf{param}[i] \mid \overline{\mathsf{return}}[i] \\ callExit[i] &\rightarrow \mathsf{return}[i] \mid \overline{\mathsf{param}}[i] \end{aligned}$$

Context Sensitive Formulation

Context-free grammer for L_c

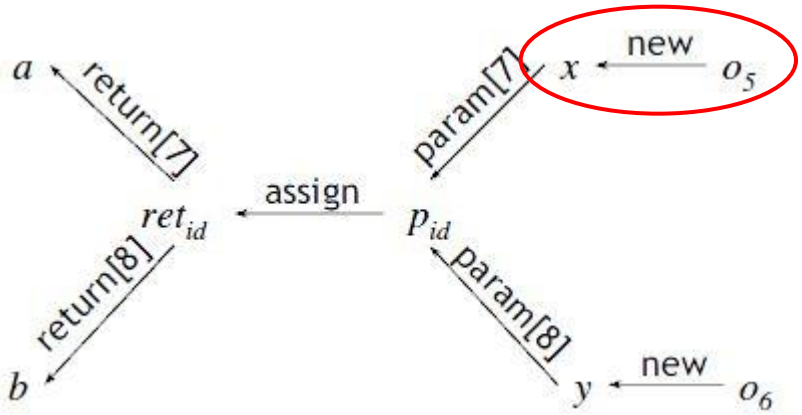
$$
\begin{aligned}
csStart &\rightarrow unbalExits\ \ unbalEntries \\
unbalExits &\rightarrow balanced\ \ unbalExits \mid callExit[i]\ \ unbalExits \mid \epsilon \\
unbalEntries &\rightarrow balanced\ \ unbalEntries \mid callEntry[i]\ \ unbalEntries \mid \epsilon \\
balanced &\rightarrow callEntry[i]\ \ balanced\ \ callExit[i] \\
&\quad \mid balanced\ \ balanced \mid nonCallTerm \mid \epsilon \\
callEntry[i] &\rightarrow \mathsf{param}[i] \mid \overline{\mathsf{return}}[i] \\
callExit[i] &\rightarrow \mathsf{return}[i] \mid \overline{\mathsf{param}}[i] \\
nonCallTerm &\rightarrow \mathsf{new} \mid \overline{\mathsf{new}} \mid \mathsf{assign} \mid \overline{\mathsf{assign}} \mid \mathsf{assignglobal} \mid \overline{\mathsf{assignglobal}} \\
&\quad \mid \mathsf{getfield}[f] \mid \overline{\mathsf{getfield}}[f] \mid \mathsf{putfield}[f] \mid \overline{\mathsf{putfield}}[f]
\end{aligned}
$$

Context Sensitive Formulation

Code Example

```
1. static Object id(Object o) {
2. return o;
3.  }
4. main() {
5.      x = new object();
6.      y = new object();
7.      a = id(x);
8.      b = id(y);
9. }
```

Graph Representation

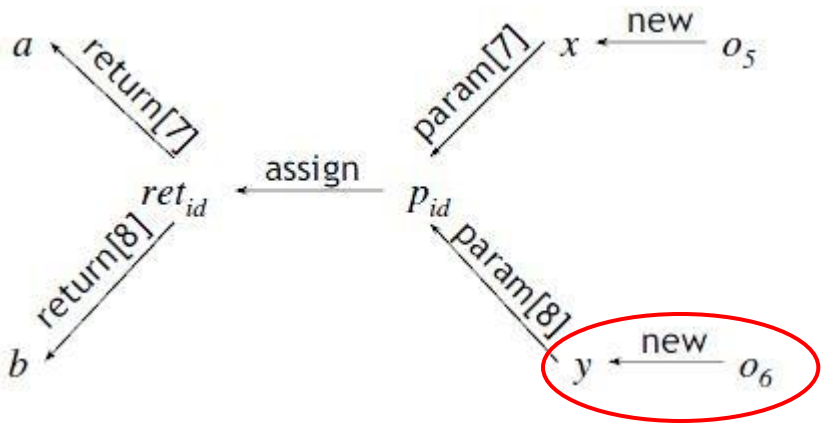


Context Sensitive Formulation

Code Example

```
1. static Object id(Object o) {
2. return o;
3.  }
4. main() {
5.      x = new object();
6.      y = new object();
7.      a = id(x);
8.      b = id(y);
9. }
```

Graph Representation

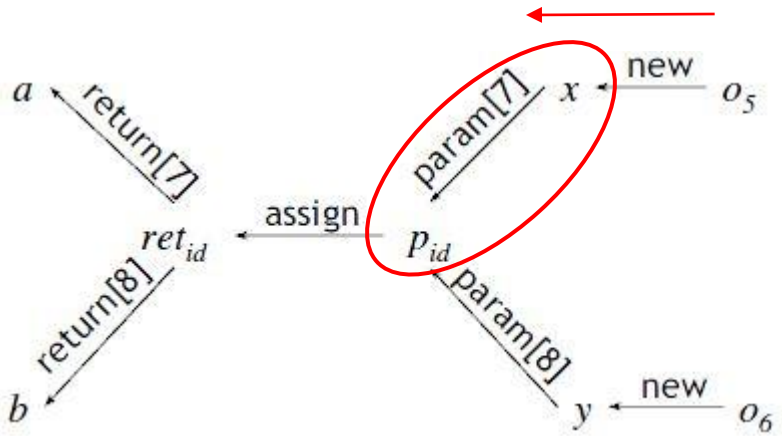


Context Sensitive Formulation

Code Example

```
1. static Object id(Object o) {
2. return o;
3.  }
4. main() {
5.      x = new object();
6.      y = new object();
7.      a = id(x);
8.      b = id(y);
9. }
```

Graph Representation

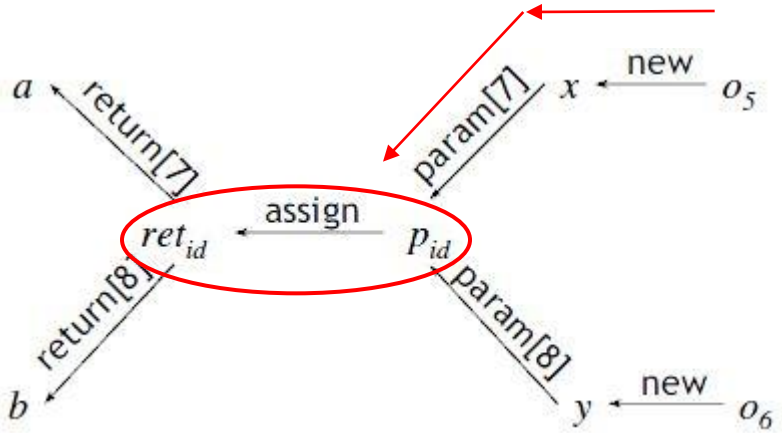


Context Sensitive Formulation

Code Example

```
1. static Object id(Object o) {
2. return o;
3.  }
4. main() {
5.      x = new object();
6.      y = new object();
7.      a = id(x);
8.      b = id(y);
9. }
```

Graph Representation

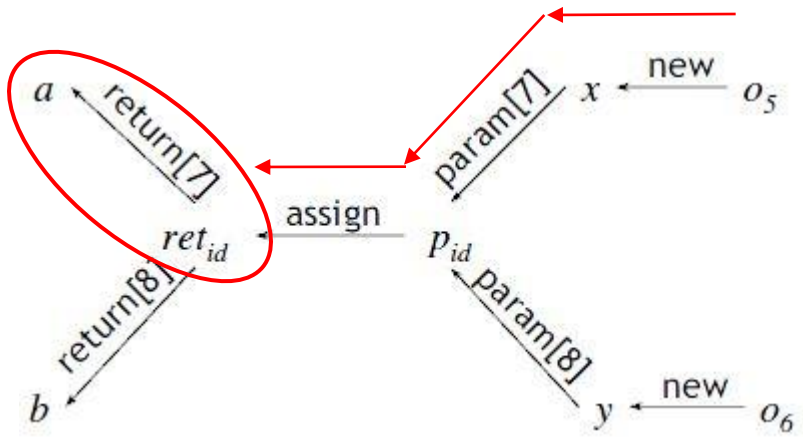


Context Sensitive Formulation

Code Example

```
1. static Object id(Object o) {
2. return o;
3.  }
4. main() {
5.      x = new object();
6.      y = new object();
7.      a = id(x);
8.      b = id(y);
9. }
```

Graph Representation

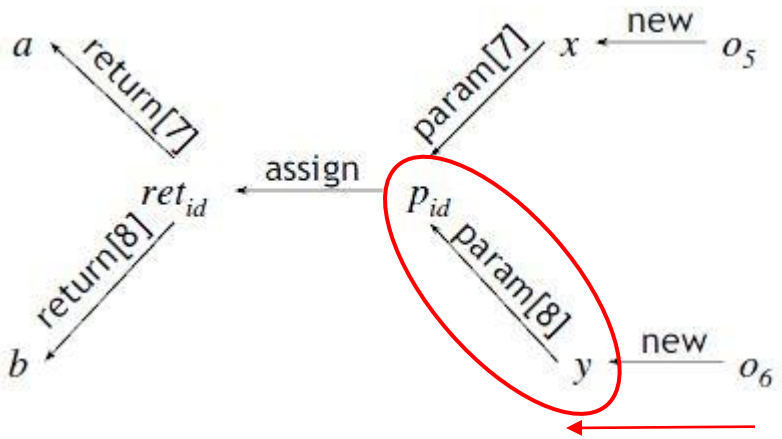


Context Sensitive Formulation

Code Example

```
1. static Object id(Object o) {
2. return o;
3. }
4. main() {
5.     x = new object();
6.     y = new object();
7.     a = id(x);
8.     b = id(y);
9. }
```

Graph Representation

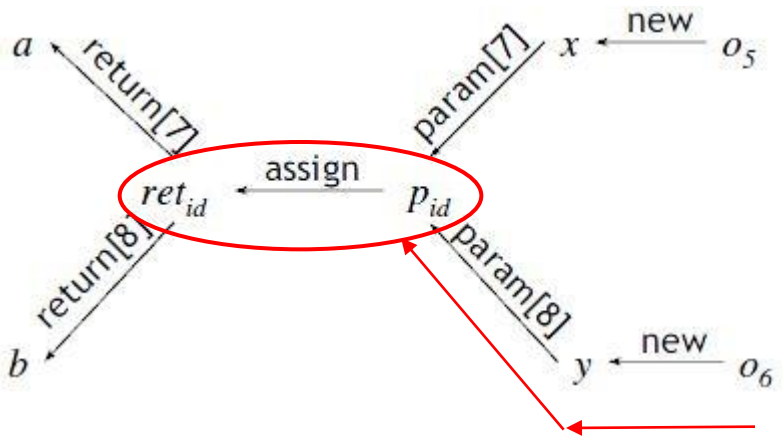


Context Sensitive Formulation

Code Example

```
1. static Object id(Object o) {
2. return o;
3.  }
4. main() {
5.      x = new object();
6.      y = new object();
7.      a = id(x);
8.      b = id(y);
9. }
```

Graph Representation

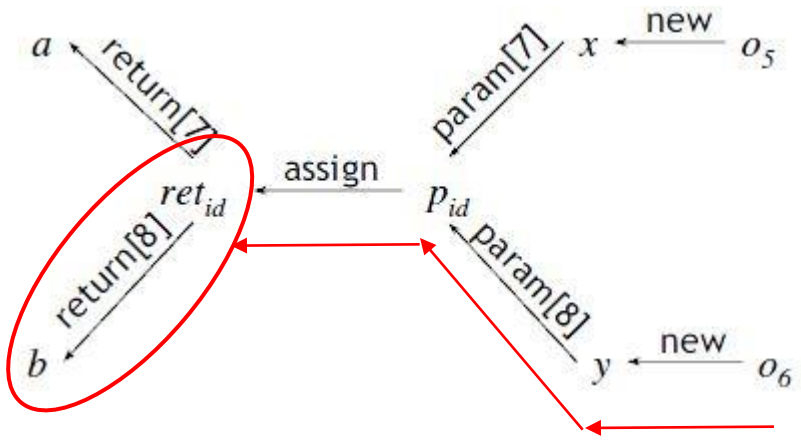


Context Sensitive Formulation

Code Example

```
1. static Object id(Object o) {
2. return o;
3. }
4. main() {
5.     x = new object();
6.     y = new object();
7.     a = id(x);
8.     b = id(y);
9. }
```

Graph Representation

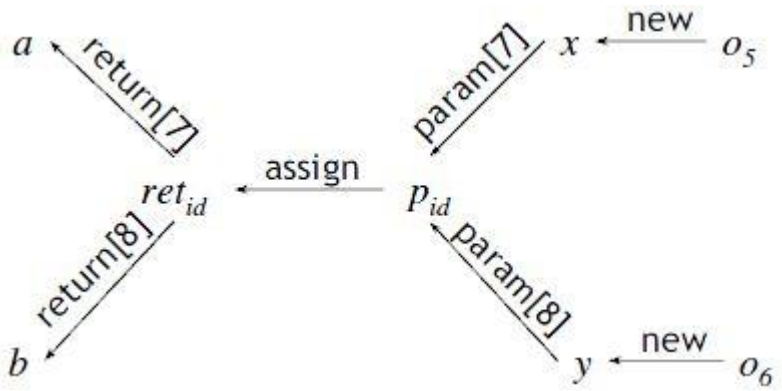


Context Sensitive Formulation

L_c Path from o_5 to a

o_5 **new** x **param[7]** p_{id} **assign** ret_{id} **return[7]** a

Graph Representation



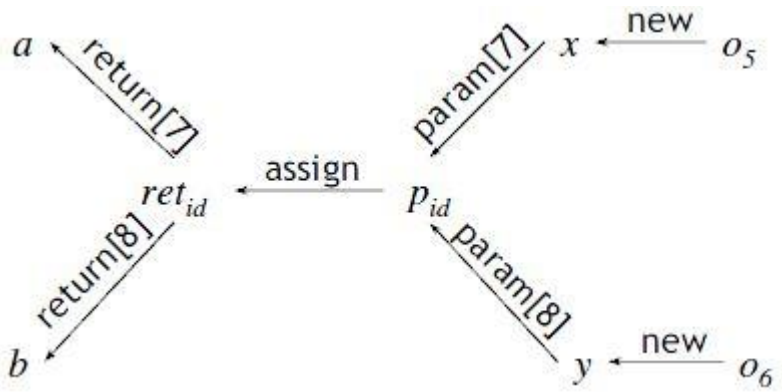
Context Sensitive Formulation

L_c Path from o_5 to a

o_5 **new** *x* **param[7]** p_{id} **assign** ret_{id} **return[7]** *a*

o_5 ***nonCallTerm*** *x* ***callEntry[7]*** p_{id} ***nonCallTerm*** ret_{id} ***callExit[7]*** *a*

Graph Representation



Context Sensitive Formulation

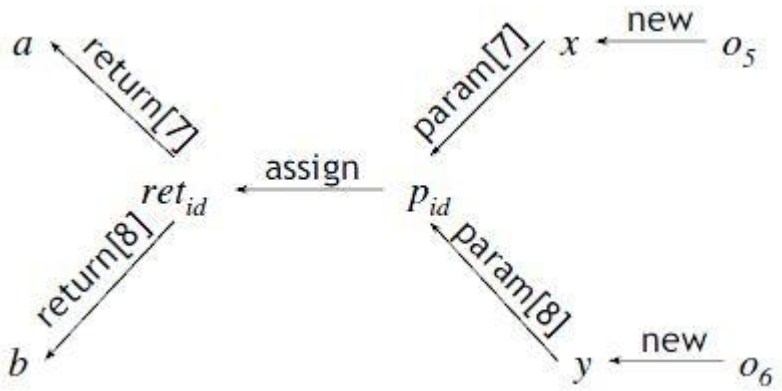
L_c Path from o_5 to a

o_5 **new** *x* **param[7]** p_{id} **assign** ret_{id} **return[7]** *a*

o_5 ***nonCallTerm*** *x* ***callEntry[7]*** p_{id} ***nonCallTerm*** ret_{id} ***callExit[7]*** *a*

o_5 ***balanced*** *x* ***balanced*** *a*

Graph Representation



Context Sensitive Formulation

L_c Path from o_5 to a

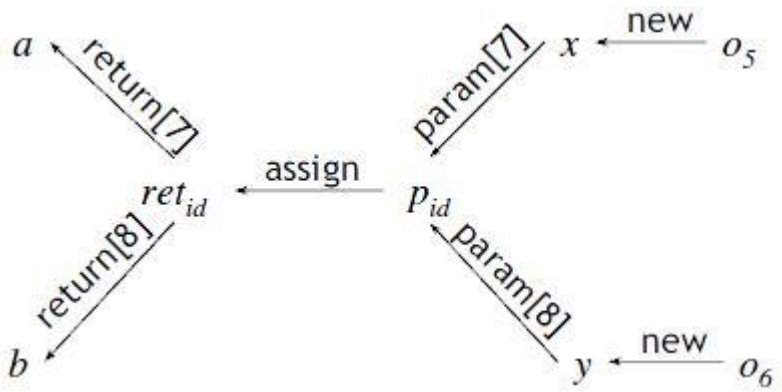
o_5 **new** *x* **param[7]** p_{id} **assign** ret_{id} **return[7]** *a*

o_5 ***nonCallTerm*** *x* ***callEntry[7]*** p_{id} ***nonCallTerm*** ret_{id} ***callExit[7]*** *a*

o_5 ***balanced*** *x* ***balanced*** *a*

o_5 ***balanced*** *a*

Graph Representation



Context Sensitive Formulation

L_c Path from o_5 to a

o_5 **new** *x* **param[7]** p_{id} **assign** ret_{id} **return[7]** *a*

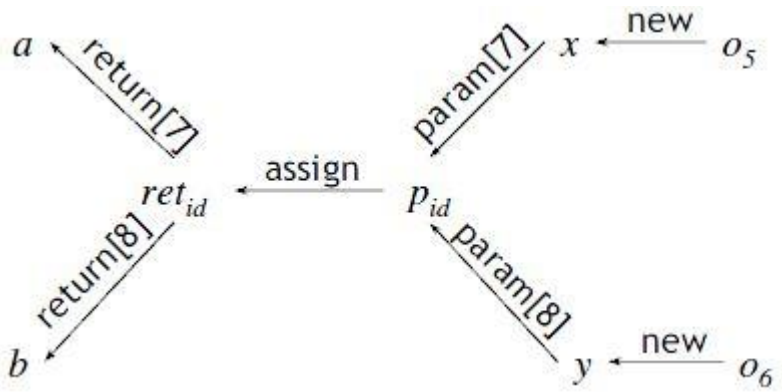
o_5 ***nonCallTerm*** *x* ***callEntry[7]*** p_{id} ***nonCallTerm*** ret_{id} ***callExit[7]*** *a*

o_5 ***balanced*** *x* ***balanced*** *a*

o_5 ***balanced*** *a*

a is **L_{cF}-reachable from** o_5

Graph Representation



Context Sensitive Formulation

L_c Path from o_5 to a

o_5 **new** *x* **param[7]** p_{id} **assign** ret_{id} **return[7]** *a*

o_5 ***nonCallTerm*** *x* ***callEntry[7]*** p_{id} ***nonCallTerm*** ret_{id} ***callExit[7]*** *a*

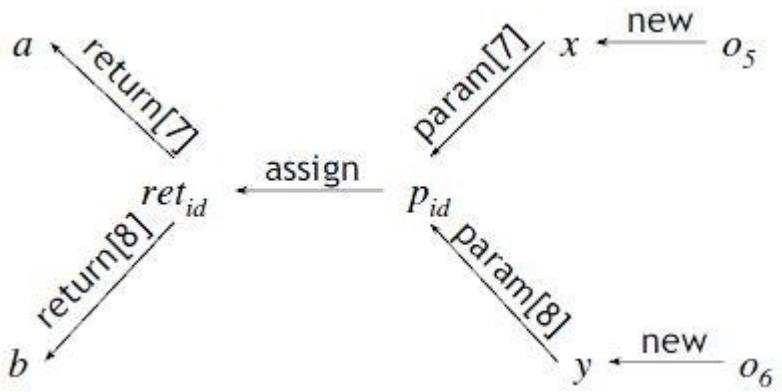
o_5 ***balanced*** *x* ***balanced*** *a*

o_5 ***balanced*** *a*

a is **L_{cF}-reachable from** o_5

No **L_c-Path from** o_5 *to b.*

Graph Representation



Context Sensitive Formulation

L_c Path from o_5 to a

o_5 **new** *x* **param[7]** p_{id} **assign** ret_{id} **return[7]** *a*

→ o_5 ***nonCallTerm*** *x* ***callEntry[7]*** p_{id} ***nonCallTerm*** ret_{id} ***callExit[7]*** *a*

→ o_5 ***balanced*** *x* ***balanced*** *a*

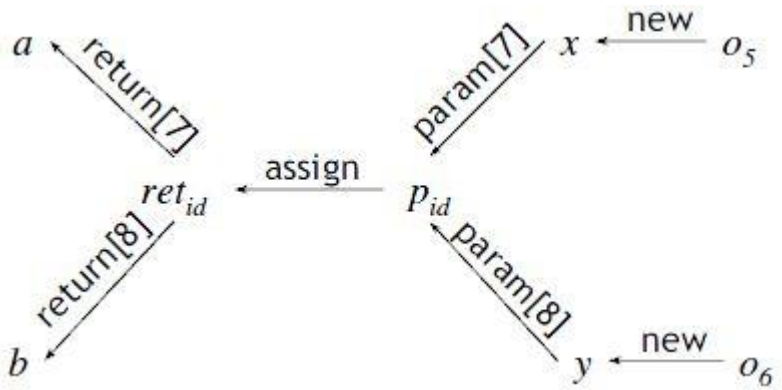
→ o_5 ***balanced*** *a*

a is **L_{cF}-reachable from** o_5

No **L_c-Path from** o_5 *to b.*

L_{cF}-reachability keeps two calls seperate by filtering out unrealizatble paths.

Graph Representation



Context Sensitive Formulation

Handling Globals

- Graph edges representing global accesses can connect locals in different methods.

Context Sensitive Formulation

Handling Globals

- Graph edges representing global accesses can connect locals in different methods.
- These edges allow paths to connect local variables in distinct methods without including the call entry and exit edges.

Context Sensitive Formulation

Handling Globals

- Graph edges representing global accesses can connect locals in different methods.
- These edges allow paths to connect local variables in distinct methods without including the call entry and exit edges.
- PROBLEM: Without modification, L_c -reachability would unsoundly filter some paths with assignglobal edges.

Context Sensitive Formulation

Handling Globals

- Graph edges representing global accesses can connect locals in different methods.
- These edges allow paths to connect local variables in distinct methods without including the call entry and exit edges.
- PROBLEM: Without modification, L_c-reachability would unsoundly filter some paths with assignglobal edges.
  - It can occur when a call entry edge precedes assignglobal edges on a path, but those assignglobal edges lead to a local in a distinct method.

Context Sensitive Formulation

Handling Globals

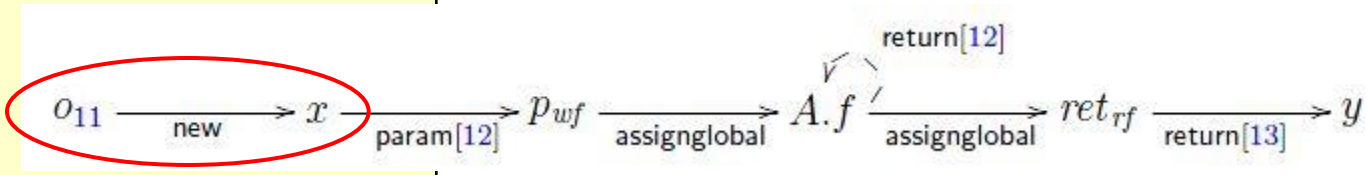
- Graph edges representing global accesses can connect locals in different methods.
- These edges allow paths to connect local variables in distinct methods without including the call entry and exit edges.
- PROBLEM: Without modification, L_c -reachability would unsoundly filter some paths with assignglobal edges.
  - It can occur when a call entry edge precedes assignglobal edges on a path, but those assignglobal edges lead to a local in a distinct method.
- SOLUTION: Add self-edges on all global variable nodes with all possible callExit[i] edges.

Context Sensitive Formulation

Code Example (Handling of Globals)

```
1.  class  A {
2.      static obj f;
3.  }
4.  obj rf() {
5.      return A.f;
6.  }
7.  void wf(obj p) {
8.      A.f = p;
9.  }
10. main () {
11.   obj x = new obj();
12.   wf(x);
13.   obj y = rf();
14. }
```

Graph Representation

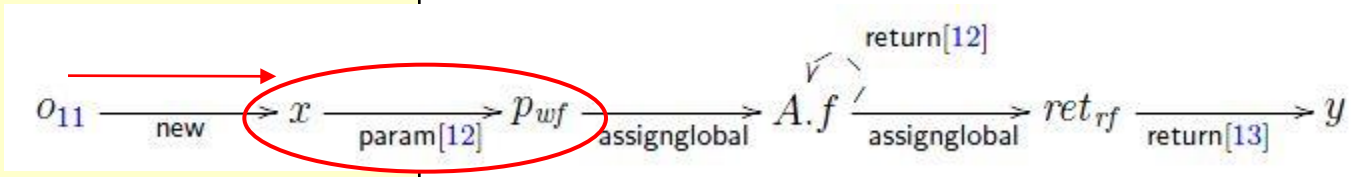


Context Sensitive Formulation

Code Example (Handling of Globals)

```
1.  class  A {
2.      static obj f;
3.  }
4.  obj rf() {
5.      return A.f;
6.  }
7.  void wf(obj p) {
8.      A.f = p;
9.  }
10. main () {
11.   obj x = new obj();
12.   wf(x);
13.   obj y = rf();
14. }
```

Graph Representation

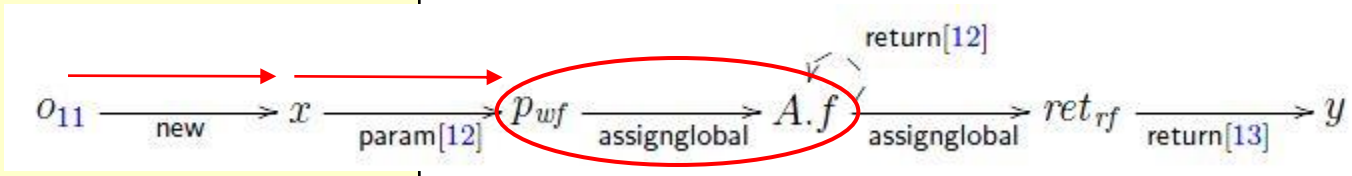


Context Sensitive Formulation

Code Example (Handling of Globals)

```
1.  class  A {
2.      static obj f;
3.  }
4.  obj rf() {
5.      return A.f;
6.  }
7.  void wf(obj p) {
8.      A.f = p;
9.  }
10. main () {
11.   obj x = new obj();
12.   wf(x);
13.   obj y = rf();
14. }
```

Graph Representation

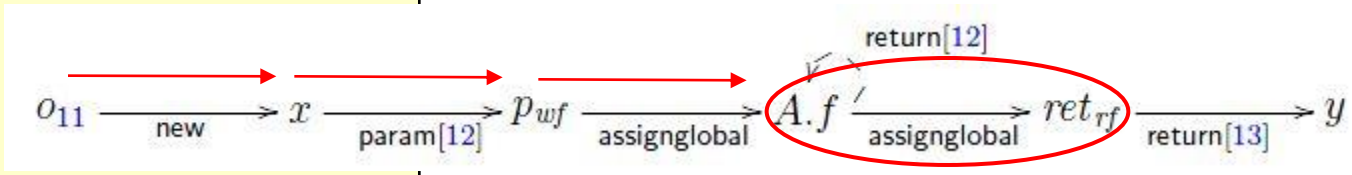


Context Sensitive Formulation

Code Example (Handling of Globals)

```
1.  class  A {
2.      static obj f;
3.  }
4.  obj rf() {
5.      return A.f;
6.  }
7.  void wf(obj p) {
8.      A.f = p;
9.  }
10. main () {
11.   obj x = new obj();
12.   wf(x);
13.   obj y = rf();
14. }
```

Graph Representation

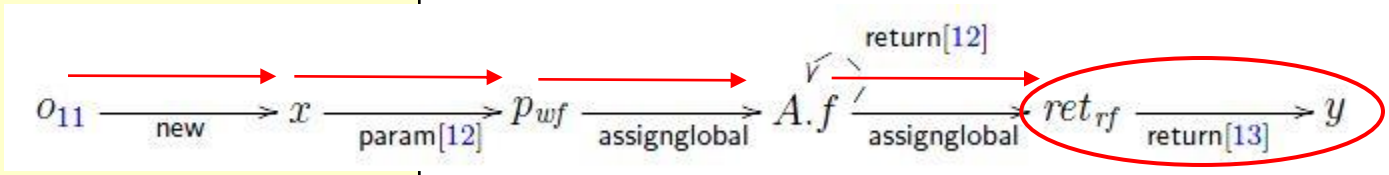


Context Sensitive Formulation

Code Example (Handling of Globals)

```
1.  class  A {
2.      static obj f;
3.  }
4.  obj rf() {
5.      return A.f;
6.  }
7.  void wf(obj p) {
8.      A.f = p;
9.  }
10. main () {
11.   obj x = new obj();
12.   wf(x);
13.   obj y = rf();
14. }
```

Graph Representation



Context Sensitive Formulation

Heap Abstraction

- L_c-reachability tracks the unmatched callEntry[i] edges on paths through abstract location nodes.

Context Sensitive Formulation

Heap Abstraction

- L_c-reachability tracks the unmatched callEntry[i] edges on paths through abstract location nodes.
- It is equivalent to creating a copy of the abstract location particular to that sequence of call entries, yielding a context-sensitive heap abstraction.

Refinement-based Context Sensitive point-to Analysis

Types of Context-sensitivity

- Filtering-out of unrealizable paths

Refinement-based Context Sensitive point-to Analysis

Types of Context-sensitivity

- Filtering-out of unrealizable paths
  - Calls and returns are matched

Refinement-based Context Sensitive point-to Analysis

Types of Context-sensitivity

- Filtering-out of unrealizable paths
  - Calls and returns are matched
- A context-sensitive heap abstraction

Refinement-based Context Sensitive point-to Analysis

Types of Context-sensitivity

- Filtering-out of unrealizable paths
  - Calls and returns are matched
- A context-sensitive heap abstraction
  - Objects allocated by same statements in different calling contexts are distinguished

Refinement-based Context Sensitive point-to Analysis

Types of Context-sensitivity

- Filtering-out of unrealizable paths
  - Calls and returns are matched
- A context-sensitive heap abstraction
  - Objects allocated by same statements in different calling contexts are distinguished
- A context sensitive call graph

Refinement-based Context Sensitive point-to Analysis

Types of Context-sensitivity

- Filtering-out of unrealizable paths
  - Calls and returns are matched
- A context-sensitive heap abstraction
  - Objects allocated by same statements in different calling contexts are distinguished
- A context sensitive call graph
  - Targets of virtual calls are computed seperately for each calling context.

Refinement-based Context Sensitive point-to Analysis

Algorithm Overview

- Fully Context and field-sensitive points-to Analysis is undecidable.
- Checking for both balanced field and method call paratheses requires reachability over the intersection of a context-free and a regular language.
- Assume presence of ahead-of-time call graph.

$$\Sigma_P = \{[_f \ , \ ]_f \mid f \text{ is a field}\} \cup \{(_i \ , \ )_i \mid i \text{ is a call site}\}$$

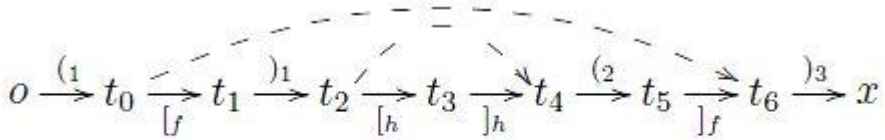
- Analysis require CFL-reachability with language $L_{scf} = L_{sf} \cap R_{sc}$ over $\sum_p$.

$$\begin{aligned} callEntry[i] &\rightarrow \text{param}[i] \mid \overline{\text{return}}[i] \\ callExit[i] &\rightarrow \text{return}[i] \mid \overline{\text{param}}[i] \end{aligned}$$

Refinement-based Context Sensitive point-to Analysis

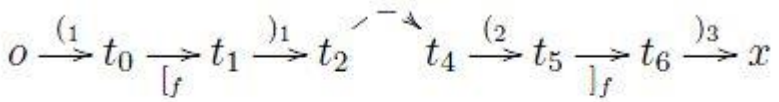
Algorithm Overview

- Paths to ilustrate the behaviour of context-sensitive refinement algorithm.



(a) Initial path with match edges.

(b) Path examined by first pass of algorithm.



(c) Path examined by second pass of algorithm.

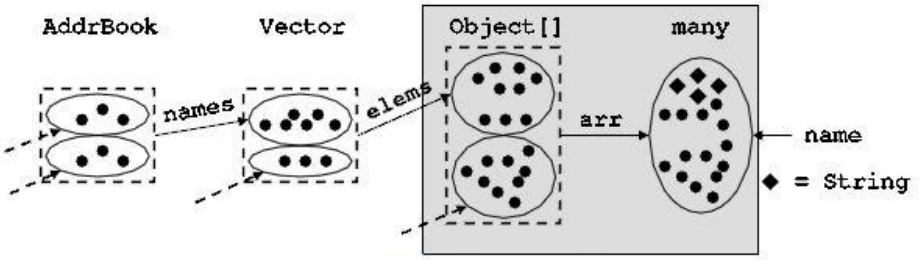
Context Sensitive Analysis

Code Example (Points to Analysis Algorithm)

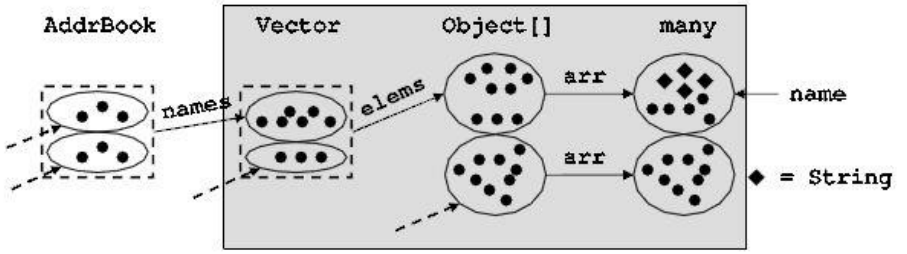
```
1  class Vector {
2     Object[] elems; int count;
3     Vector() { t = new Object[10];
4     this.elems = t; }
5     void add(Object p) {
6        t = this.elems;
7        t[count++] = p; // writes t.arr
8     }
9     Object get(int ind) {
10       t = this.elems;
11       return t[ind]; // reads t.arr
12    } ...
13 }
14 class AddrBook {
15    private Vector names;
16    AddrBook() { t = new Vector();
17    this.names = t; }
18    void addEntry(String n, ...) {
19          t = this.names; ...;
20          t.add(n);
21.   }
```

```
22    void update() {
23       t = this.names;
24       for (int i = 0; i < t.size(); i++) {
25            Object name = t.get(i);
26          // is this cast safe?
27    String nameStr = (String)name;
28     ...
29     }
30    }
31 }
32 void useVec() {
33       Vector v = new Vector();
34       Integer i1 = new Integer();
35       v.add(i1);
36       Integer i2 = (Integer)v.get(0);
37 }
```

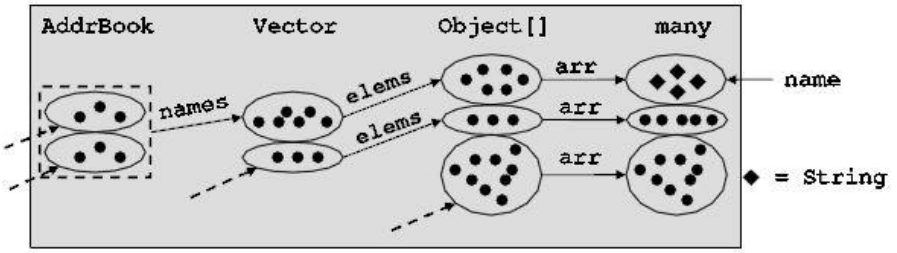
Graph Representation



(a) Initial analysis result.



(b) Result after distinguishing `Object[]` contents.



(c) Result after also distinguishing `Vector` contents.

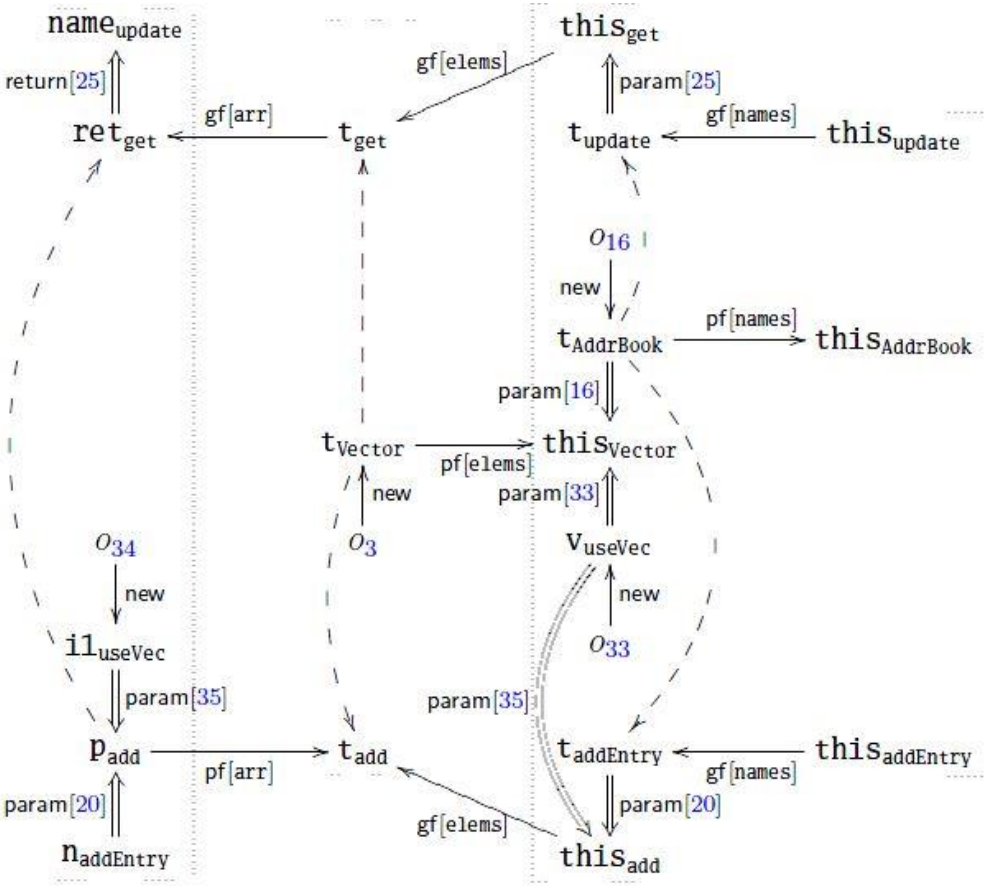
Context Sensitive Analysis

Code Example (Points to Analysis Algorithm)

```
1  class Vector {
2     Object[] elems; int count;
3     Vector() { t = new Object[10];
4     this.elems = t; }
5     void add(Object p) {
6        t = this.elems;
7        t[count++] = p; // writes t.arr
8     }
9     Object get(int ind) {
10       t = this.elems;
11       return t[ind]; // reads t.arr
12    } ...
13 }
14 class AddrBook {
15    private Vector names;
16    AddrBook() { t = new Vector();
17    this.names = t; }
18    void addEntry(String n, ...) {
19          t = this.names; ...;
20          t.add(n);
21.   }
```

```
22    void update() {
23       t = this.names;
24       for (int i = 0; i < t.size(); i++) {
25             Object name = t.get(i);
26          // is this cast safe?
27    String nameStr = (String)name;
28     ...
29     }
30    }
31 }
32 void useVec() {
33       Vector v = new Vector();
34       Integer i1 = new Integer();
35       v.add(i1);
36       Integer i2 = (Integer)v.get(0);
37 }
```

Graph Representation



Conclusion

- Following it with On-The-Fly Call graph

References

- Aho A.V., Sethi R. & Ullman J.D. “Compilers: Principles, Techniques and Tools”, Addison Wesley.
- Sridharan M., Bodík R. “Refinement based Context-Sensitive points-to Analysis for java” PLDI ’06.
- Sridharan, Manu. “Refinement based program analysis tools” Dissertation. University of California at Berkeley, 2007.