Field-Sensitive Points-to-Analysis

Eda GÜNGÖR
s6edguen@uni-bonn.de

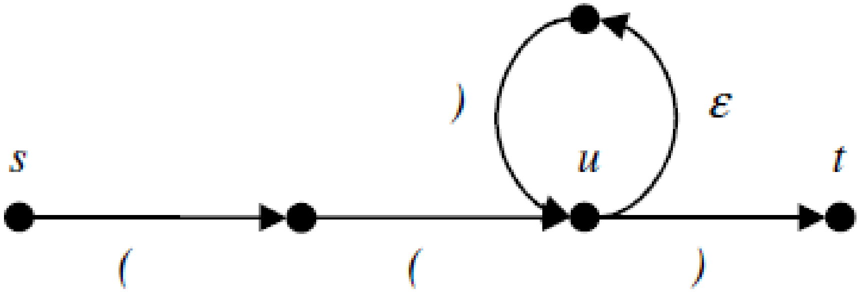
Outline

- CFL-reachability
- Aim of the PointsTo Analysis
- Field-sensitive Versus Field-based
- Context-Insensitive Formulation
  - Graph representation
  - Analysis Grammar
- Context-Insensitive Points-To Analysis
  - Refinement algorithm
  - Regular Approximation
  - Refinement with Regular Reachability
  - Evaluation

What is CFL-reachability?

- Context-free language reachability
- An extension of traditional graph reachability
- Interesting paths in G are described by the language “L”
- The CFL-reachability problem requires determining for all node pairs, if there is an L-path between these nodes

CFL-reachability with an Input Graph



- Each path p has a string s(p), constructed by edge labels
  - s(p) = “(())”
- p is an L-path iff s(p) Є L
  - There is an L-path from s to t
  - There is not any L-path from u to t
- t is L-reachable from s (simply, s L t)
  - t is not L-reachable from u

The Aim of the PointsTo Analysis

- Computing the best possible (most precise) flow-insensitive points-to information
  - Flow-insensitive: treats each method as if its control-flow graph contains all possible edges
  - Requires a subset-based, field-sensitive manner
  - Input program with
    - No arrays
    - No recursive method calls

Field-sensitive versus Field-based

```
x = new Obj();      // o1
y = new Obj();      // o2
x.f = new Obj();    // o3
y.f = new Obj();    // o4
z = x.f;            // pt(z)=?
```

pt(z): points-to set of variable z

Field-sensitive analysis: Reasons separately about the instance fields of each abstract object. Since, x and y cannot be aliased, the result is pt(z) = {o3}

Field-based analysis: Treats each instance field as a global variable, yielding pt(z) = {o3, o4}

Context-Insensitive Formulation

Graph Representation

Analysis Grammar

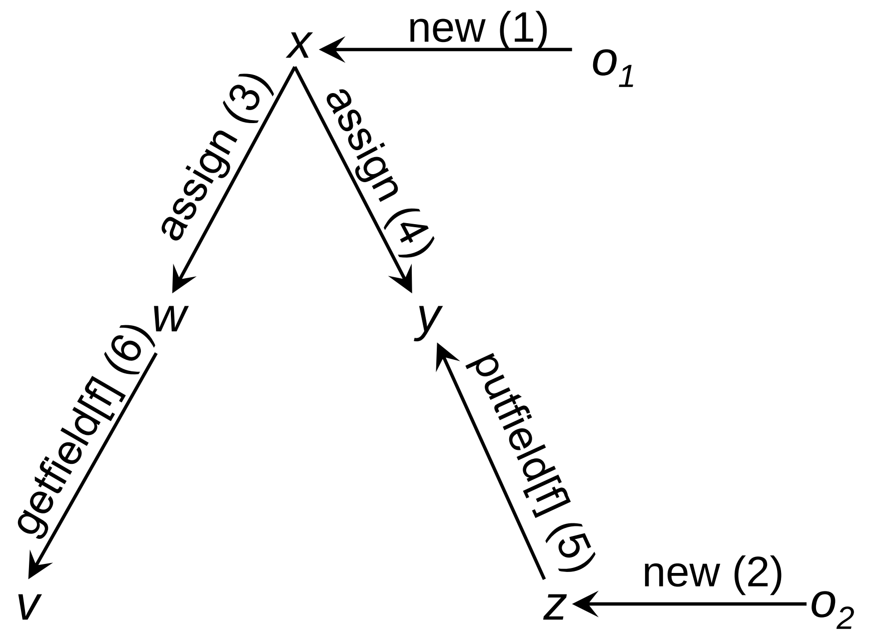
Graph Representation

Statement		Graph Edge	
s: $x = \text{new } T()$	$\Rightarrow$	$o_s \xrightarrow{\text{new}} x$	(allocation statements)
$x = y$	$\Rightarrow$	$y \xrightarrow{\text{assign/assignglobal}} x$	(copy statements)
$x = y.f$	$\Rightarrow$	$y \xrightarrow{\text{getfield[f]}} x$	(heap reads)
$x.f = y$	$\Rightarrow$	$y \xrightarrow{\text{putfield[f]}} x$	(heap writes)
s: $x = m(a_1, a_2, \ldots, a_k)$	$\Rightarrow$	$a_i \xrightarrow{\text{param[s]}} f_{m,i}$	(for $i \in [1,2,\ldots,k]$)
		$ret_m \xrightarrow{\text{return[s]}} x$	

- Value flow of the statement from right-hand side to left-hand side
- The field name f is part of the edge label

A Code Example and Its Graph Representation

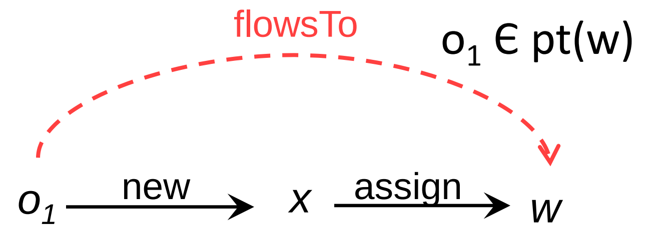
```
1   x = new Obj();  //o1
2   z = new Obj();  //o2
3   w = x;
4   y = x;
5   y.f = z;
6   v = w.f;
```



Grammar of the Analysis

- The language L_F is defined by the grammar

  $$\text{flowsTo} \rightarrow \text{new (assign)}^*$$



- x is L_F-reachable from o iff o Є pt(x)

- To track value flow through the heap via putfield[f] and getfield[f] statements, L_F is extended

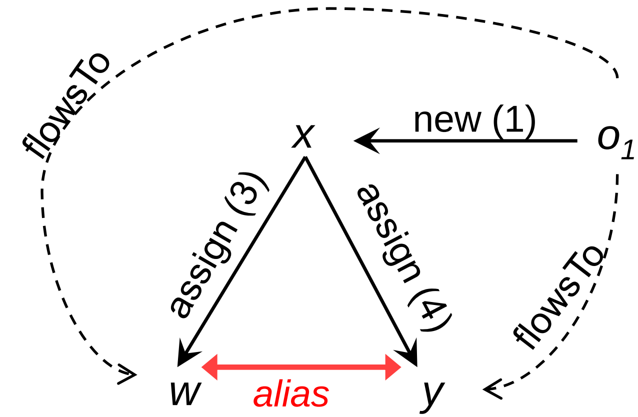
- Extended flowsTo grammar

  flowsTo → new (assign | putfield[f] *alias* getfield[f])*

May-aliasing

- Aliasing by the means of flowsTo-paths:
  - x and y are **may-aliased** iff
    - o flowsTo x
    - o flowsTo y
    - o Є pt(x) ∩ pt(y)

    for an object o



Problem:

- May-aliasing is unsuitable for CFL-reachability since it can only check language membership of a single path

Solution:

- Extending the graph representation to allow for <u>inverse paths</u>

Inverse Paths

- Enable alias paths by introducing reversed flowsTo paths ($\overline{\text{flowsTo}}$)
- Connect two may-aliased variables and allow for a single path
- The grammar:

$$\mathit{alias} \rightarrow \overline{\text{flowsTo}}\ \ \text{flowsTo}$$

$$\overline{\text{flowsTo}} \rightarrow (\overline{\text{assign}} \mid \overline{\text{putfield}}[\text{f}]\ \mathit{alias}\ \overline{\text{getfield}}[\text{f}])^{*}\ \overline{\text{new}}$$

- An (x alias y)-path can be defined as a path

  x $\overline{\text{flowsTo}}$ n flowsTo y (for some node n)

A Context-free Grammar for L_F

$$
\begin{aligned}
flowsTo &\rightarrow \mathsf{new} \\
\overline{flowsTo} &\rightarrow \overline{\mathsf{new}} \\
flowsTo &\rightarrow flowsTo\ \ ciAssign \\
\overline{flowsTo} &\rightarrow \overline{ciAssign}\ \ \overline{flowsTo} \\
flowsTo &\rightarrow flowsTo\ \ \mathsf{putfield}[f]\ \ alias\ \ \mathsf{getfield}[f] \\
\overline{flowsTo} &\rightarrow \overline{\mathsf{getfield}}[f]\ \ alias\ \ \overline{\mathsf{putfield}}[f]\ \ \overline{flowsTo} \\
alias &\rightarrow \overline{flowsTo}\ \ flowsTo \\
ciAssign &\rightarrow \mathsf{assign} \mid \mathsf{assignglobal} \mid \mathsf{param}[i] \mid \mathsf{return}[i] \\
\overline{ciAssign} &\rightarrow \overline{\mathsf{assign}} \mid \overline{\mathsf{assignglobal}} \mid \overline{\mathsf{param}}[i] \mid \overline{\mathsf{return}}[i] \\
pointsTo &\rightarrow \overline{flowsTo}
\end{aligned}
$$

A Context-free Grammar for L_F

- Each $\overline{\text{flowsTo}}$ inverts the edges of flowsTo production
- The ciAssign and $\overline{\text{ciAssign}}$ non-terminals treat edges
- Determining a points-to set for a variable x requires solving a backwards L_F-reachability problem from x
- pointsTo → $\overline{\text{flowsTo}}$ production makes this backwards-reachability correspondence explicit:
  - o Є pt(x) iff x pointsTo o (x $\overline{\text{flowsTo}}$ o)

Example: flowsTo-path from o_2 to v

y $\overline{\text{assign}}$ x $\overline{\text{new}}$ o_1 new x assign w

→ y $\overline{\text{ciAssign}}$ x $\overline{\text{flowsTo}}$ o_1 flowsTo x ciAssign w

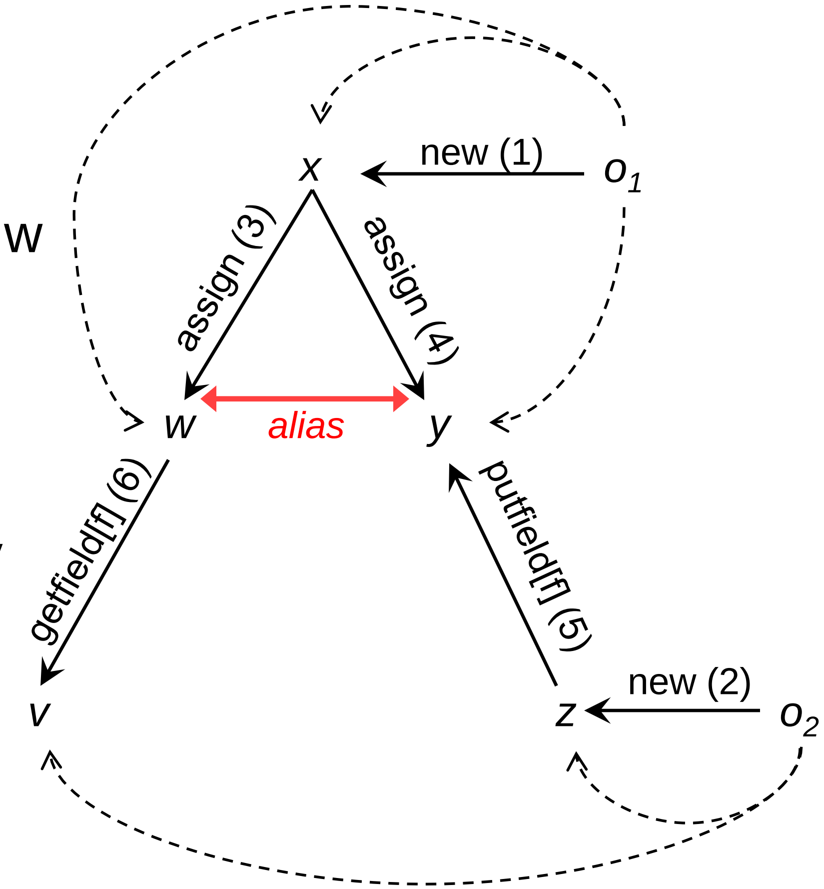
→ y $\overline{\text{flowsTo}}$ o_1 flowsTo w

→ y alias w

→ o_2 new z putfield[f] y alias w getfield[f] v

→ o_2 flowsTo z putfield[f] y alias w getfield[f] v

→ o_2 flowsTo v



Context-Insensitive Points-To Analysis

Refinement Algorithm

Regular Approximation

Refinement with Regular Reachability

Evaluation

Refinement-based Algorithm

- Solves the Lsf-reachability, simplified Lf-reachability, problem
- Quickly proves that a node is not Lsf-reachable from another node
- Focuses on unbalanced parentheses in the graph and skipping over the rest of it
- Improves performance through “approximation” and “refinement”
  - **Approximate analysis:** must answer correctly when p is an Lsf-path, but can answer incorrectly when it is not
  - **Refinement:** gradually removes the imprecision of this analysis, eventually yielding the correct answer when p is not an Lsf-path.
- To maintain correctness, the algorithm only skips sub-paths beginning and ending with matched parentheses

Regular Approximation

- Computes only the initial approximation of the refinement algorithm
  - i.e., reachability over a graph with all possible match edges
- Approximation requires regular reachability over a regular language R_F
- Asymptotically less expensive than CFL-reachability
- Has a simple and efficient demand-driven algorithm RegularPT to find points-to information based on RF-reachability
  - A demand-driven: determining which nodes are L-reachable from some specific source node

Regular Reachability

- A regular approximation of L_F-reachability via match edges
- flowsTo path for L_F

$$\text{flowsTo} \rightarrow \text{new (assign | putfield[f] } \textit{alias} \text{ getfield[f])*}$$

- LF-reachability is checking putfield[f] and getfield[f] edges (the balanced parentheses of LF) for the field-sensitivity conditions
- The most expensive part of field-sensitivity
  - Checking for an alias-path between the base variables of putfield[f] and getfield[f] edges
- **Match edges:** are used instead of "putfield[f] *alias* getfield[f]" path
- R_F is defined by the grammar

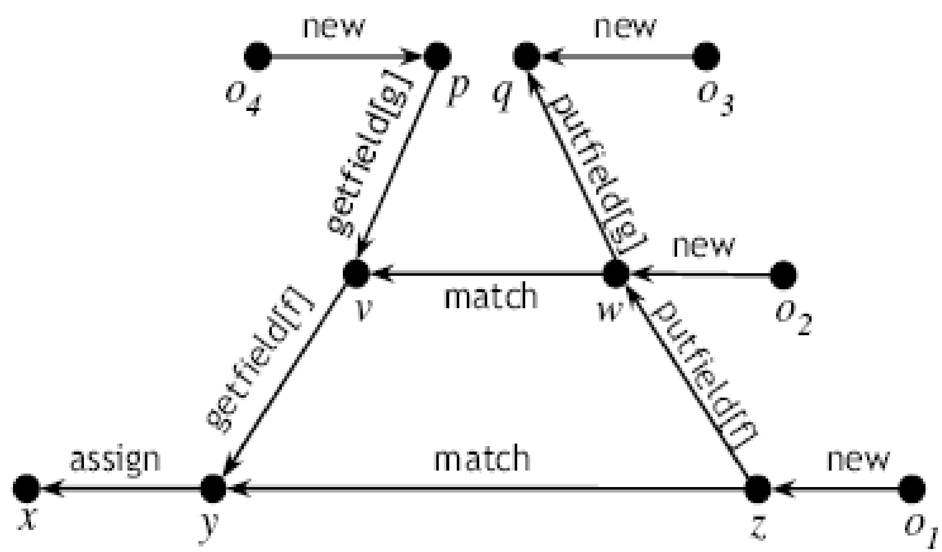
$$\text{flowsTo} \rightarrow \text{new (assign | match)*}$$

A Graph with All Possible Match Edges

- Match edges:

  from the source of putfield[f] edge

  to the target of getfield[f] edge
- v is R_F-reachable from o_2 but not L_F-reachable,
  - Since, there is no alias-path from q to p
- Match edges are added without checking the base variables are connected by an alias-path or not



Results:

- R_F-reachability over-approximates L_F-reachability
- Precision is lost in cases where an R_F-path includes an invalid match edge

Refinement with Regular Reachability

- Lost of precision can be recovered by Refinement technique
- Has an algorithm RefinedRegularPT, an extension of the RegularPT
- Removes match edges by checking the existence of an alias-path between the base variables of the corresponding putfield[f] and getfield[f] edges
- Instead of searching for alias-paths, refines a match edge by looking for aliasReg-paths, defined as

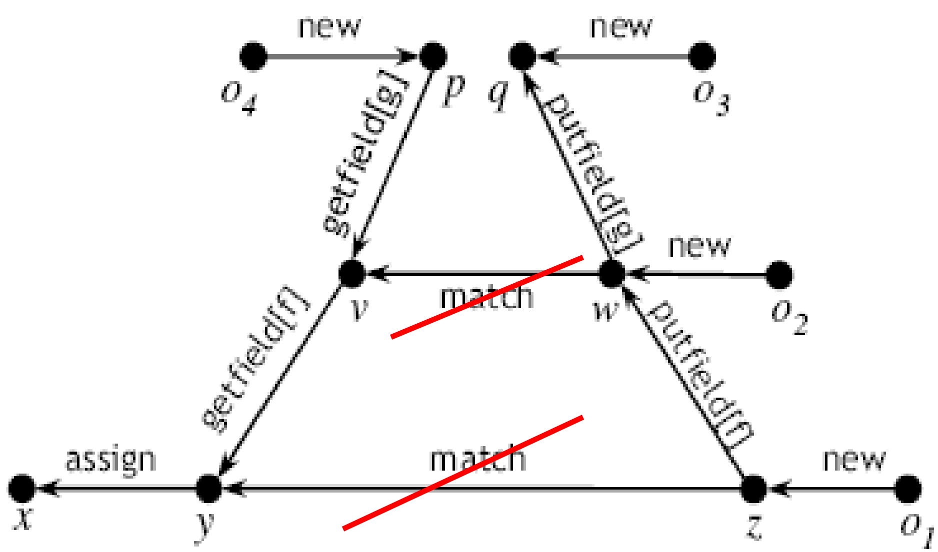
$$aliasReg \rightarrow \overline{flowsToReg}\ flowsToReg$$

$$\overline{flowsToReg} \rightarrow (\ \overline{\mathsf{assign}}\ \mid\ \overline{\mathsf{match}}\ )^{*}\ \ \overline{\mathsf{new}}$$

Example: Removing invalid match edges

Refinement by removing invalid match edges

1. Search for an aliasReg-path from q to p
2. No such path exists
3. Match edge w → v can be safely removed from the graph
4. This removal eliminates the aliasReg-path from w to v
5. Match edge z → y can also be safely removed



Evaluation: Benchmark Results

Benchmark	Intra (Live)	Reg (Live)	RefReg (Live)
soot	18.4 (16.0)	94.1 (98.5)	96.9 (99.8)
compress	26.0 (23.1)	89.1 (96.4)	93.7 (98.9)
jess	25.4 (22.5)	89.4 (96.6)	93.9 (99.0)
raytrace	26.1 (23.1)	89.1 (96.4)	93.7 (98.9)
db	25.7 (22.7)	89.3 (96.5)	93.7 (98.9)
javac	25.3 (22.3)	89.9 (96.7)	94.1 (98.8)
mpeg	26.0 (23.1)	89.1 (96.4)	94.4 (98.9)
jack	27.0 (25.1)	91.8 (97.5)	95.2 (99.2)

- Percentage results of measuring the precision of the algorithms
  - Intra: An intraprocedural field-based analysis
  - Reg: RegularPT
  - Refreg: RefinedRegularPT with a 5 second time limit per query

Conclusions

- The algorithms are precise
  - RegularPT and RefinedRegularPT have nearly precision of field-sensitive Andersen's.
  - RefinedRegularPT provides more precision than RegularPT
- Precision is retained under early termination
  - RegularPT and RefinedRegularPT retain almost all their precision when they run with small time budgets
- Good performance

References

- Refinement-Based Program Analysis Tools, by *Sridharan,* 2007, University of California, Berkeley
  http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-125.html

- Refinement-Based Context-Sensitive Points-To Analysis for Java, by Sridharan and Bodík*,* 2006, University of California, Berkeley

Thank you for your attention!