Field-sensitive Points-to Analysis with CFL-Reachability

Author: Vladimir Kutuev

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg State University

17.12.2021

CFL-Reachability

- Context-free grammar — $G = (\Sigma, N, P, S)$
  - Σ — set of terminal symbols
  - N — set of non-terminal symbols
  - P — set of production rules $\{A \to \alpha | A \in N, \alpha \in (\Sigma \cup N)^*\}$
  - $S \in N$ — start symbol
- Graph — $\mathcal{G} = (V, E, l)$
  - V — set of vertices
  - $E \subseteq V \times V$ — set of edges
  - $l : E \to L$
- $\{(v_1, v_n) : \exists p = (e_1, e_2, ..., e_n)) \in E^*, src(e_1) = v_1, dst(e_n) = v_n, l(e_1) \cdot ... \cdot l(e_n) \in L(G)\}$

Field-sensitive Points-to Analysis

- Graph

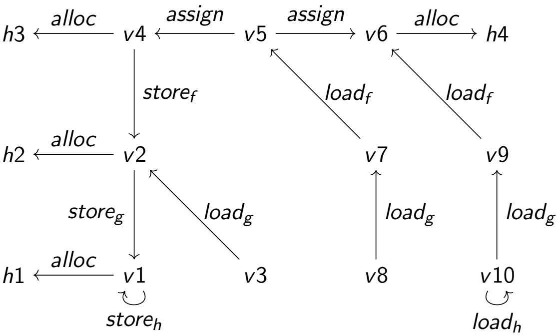
Relation	Statement	Graph edge
$alloc \subseteq \text{Vars} \times \text{Objects}$	x = new Obj();	$x \xrightarrow{alloc} h$
$assign \subseteq \text{Vars} \times \text{Vars}$	x = y;	$x \xrightarrow{assign} y$
$load_f \subseteq \text{Vars} \times \text{Vars}$	x = y.f;	$x \xrightarrow{load_f} y$
$store_f \subseteq \text{Vars} \times \text{Vars}$	x.f = y;	$x \xrightarrow{store_f} y$

- Context-free grammar defining analysis

$$
\begin{aligned}
PointsTo &\rightarrow (assign \mid load_f \; Alias \; store_f)^* \; alloc \\
Alias &\rightarrow PointsTo \; \overline{FlowsTo} \qquad\qquad \forall \text{ f} \in Fields \\
FlowsTo &\rightarrow \overline{alloc} \; (\overline{assign} \mid \overline{store_f} \; Alias \; \overline{load_f})^*
\end{aligned}
$$

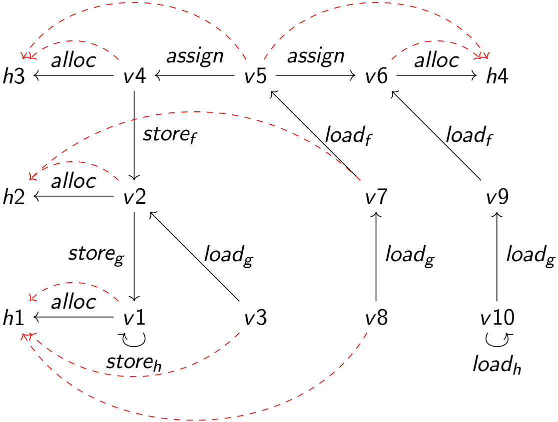
Example Graph

```
v1 = new Obj(); // h1
v2 = new Obj(); // h2
v4 = new Obj(); // h3
v6 = new Obj(); // h4
v5 = v4;
v5 = v6;
v1.h = v1;
v2.g = v1;
v4.f = v2;
v7 = v5.f;
v9 = v6.f;
v3 = v2.g;
v8 = v7.g;
v10 = v9.g;
v10 = v10.h;
```



Example Graph with PointsTo Edges

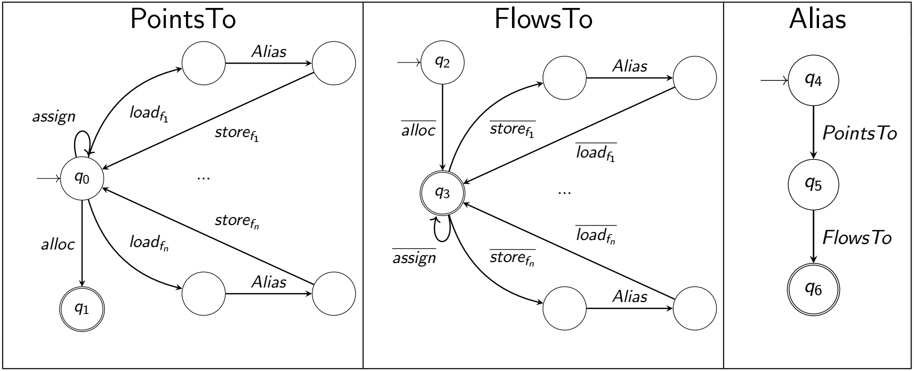
```
v1 = new Obj(); // h1
v2 = new Obj(); // h2
v4 = new Obj(); // h3
v6 = new Obj(); // h4
v5 = v4;
v5 = v6;
v1.h = v1;
v2.g = v1;
v4.f = v2;
v7 = v5.f;
v9 = v6.f;
v3 = v2.g;
v8 = v7.g;
v10 = v9.g;
v10 = v10.h;
```



Algorithms Based on Linear Algebra Operations

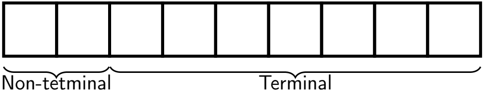
- Matrix multiplication based algorithm
  - Grammar should be converted to a Weak Chomsky Normal Form which significantly increases its size
  - The performance of the algorithm depends on the grammar size
- Kronecker Product based algorithm
  - Grammar should be converted to a recursive state machine (RSM)
  - For the intersection of RSM and a directed labeled graph, they are represented as a composition of adjacency boolean matrices for each label

RSM for Field-sensitive Points-to Analysis



One Terminal, One Non-terminal Representation

- The graph and RSM are represented as adjacency matrices, whose elements are integers
- The high-order bits encode the non-terminal, the low-order bits encode the terminal



- Operation for Kronecker product
  - $times(x, y) = (x_{nonterm} = y_{nonterm} \neq 0 \ or \ x_{term} = y_{term} \neq 0)$

Adapting Graph to this Representation

Edges	New variable	Result
$\begin{tikzcd} x \arrow[r, "assign"'] \arrow[r, bend left, "\dots"] & y \end{tikzcd}$	z = y; x = z;	$\begin{tikzcd} x \arrow[r, "assign"] \arrow[rr, bend left, "\dots"] & z \arrow[r, "assign"] & y \end{tikzcd}$
$\begin{tikzcd} x \arrow[r, "store_f"'] \arrow[r, bend left, "\dots"] & y \end{tikzcd}$	z = y; x.f = z;	$\begin{tikzcd} x \arrow[r, "store_f"] \arrow[rr, bend left, "\dots"] & z \arrow[r, "assign"] & y \end{tikzcd}$
$\begin{tikzcd} x \arrow[r, "load_f"'] \arrow[r, bend left, "\dots"] & y \end{tikzcd}$	z = y.f; x = z;	$\begin{tikzcd} x \arrow[r, "assign"] \arrow[rr, bend left, "\dots"] & z \arrow[r, "load_f"] & y \end{tikzcd}$

Results

- The algorithm based on the Kronecker product has been optimized for the field-sensitive points-to analysis problem
- The developed algorithm was implemented within the CFPQ_PyAlgo[1] framework
- The correctness of the algorithm has been checked on the Dataset[2]

Future Work:

- Extend Benchmark Graphalytics[3]
  - Investigate the performance of the developed algorithm
  - Compare performance with other CFL-Reachability algorithms

[1] https://github.com/JetBrains-Research/CFPQ_PyAlgo

[2] https://bitbucket.org/jensdietrich/gigascale-pointsto-oopsla2015

[3] https://graphalytics.org/