Context-sensitive Pointer Analysis with Cycle Elimination and Hash-consing

Woongsik Choi

KAIST

ROSAEC 3rd Workshop

2010. 1. 7

Cycle Elimination

- $\mathcal{X}_1 \supseteq \mathcal{X}_2 \supseteq \cdots \supseteq \mathcal{X}_n \supseteq \mathcal{X}_1$
  - All $\mathcal{X}_1 \sim \mathcal{X}_n$ have same solution
  - They can be merged into single representative
  - No precision is lost
- Pointer analysis
  - New edges are added as more pointer values get known
  - Dynamic transitive closure
- Online cycle elimination for pointer analysis
  - Many approaches for context-insensitive pointer analysis
  - No attempt for context-sensitive pointer analysis

Cloning-based Context-sensitivity (1/3)

- Get analysis result same as fully inlined program
- Infinite for recursive program
  - No inlining for recursive calls
  - Exponential but finite
- Fully Binary Decision Diagram (BDD) based approach
  - "Cloning-Based Context-sensitive Pointer Alias Analysis using BDD" by Whaley and Lam
  - Redundancy sharing through BDD
  - Encode contexts *and* constraints with BDD
  - Not ideal for cycle elimination

Cloning-based Context-sensitivity (2/3)

- Fully BDD-based approach

$$\frac{\mathcal{X} \supseteq_c \mathcal{Y} \qquad \mathcal{Y} \supseteq_c a}{\mathcal{X} \supseteq_c a}$$

  - Separate constraint for each context
  - Exponential constraints are represented by BDD

- Restrict sharing technique only for context

$$\frac{\mathcal{X} \supseteq_{\rho_1} \mathcal{Y} \qquad \mathcal{Y} \supseteq_{\rho_2} a}{\mathcal{X} \supseteq_{\rho_1 \sqcap \rho_2} a} \ \rho_1 \sqcap \rho_2 \neq \bot$$

  - Explicit set of context
  - BDD-encoded context is not a must

Cloning-based Context-sensitivity (3/3)

- Context-sensitivity only for address-not-taken locals
  - Exclusively belong to a function
  - Inter-procedural effect only through parameter, return

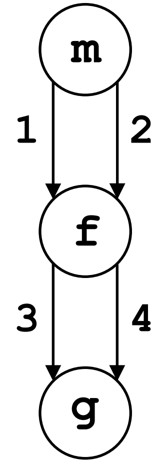
$$\frac{\mathcal{X} \supseteq_{\rho_1} \mathcal{Y} \qquad \mathcal{Y} \supseteq_{\rho_2} a}{\mathcal{X} \supseteq_{\top} a} \; \neg cs(\mathcal{X}) \; \wedge \; \rho_1 \sqcap \rho_2 \neq \bot$$

Invocation Graph

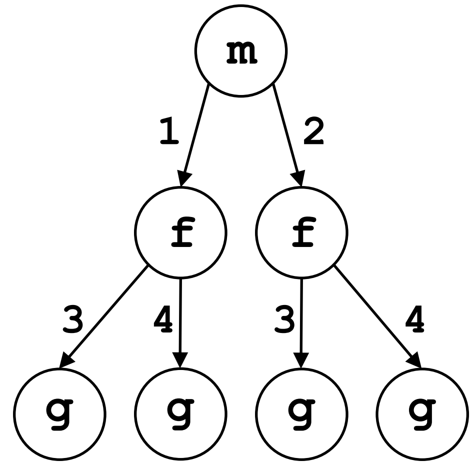
Program

```
void m() {
   f()[1];
   f()[2];
}

void f() {
   g()[3];
   g()[4];
}

void g() { }
```

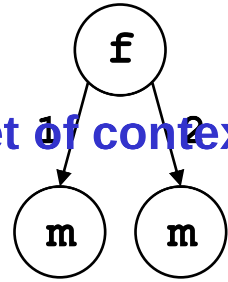
Call graph



Invocation graph

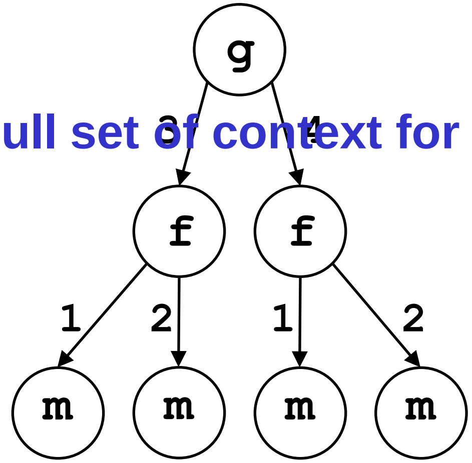


Reversed graph for `f`



Full set of context for `f`

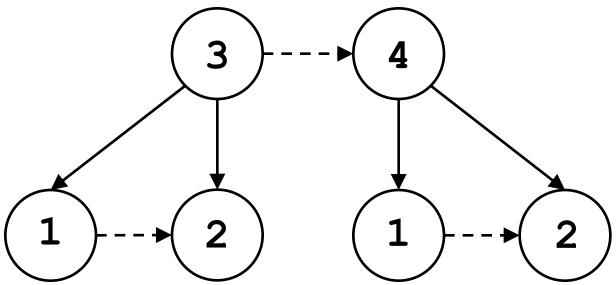
Reversed graph for `g`



Full set of context for `g`

Context Representation

- First child/next sibling binary tree
  - Encodes arbitrary forest



- Sort sibling on node label
  - Siblings represent union of contexts

- Explicit $\top$ for full set of context
  - $\top$ is polymorphic over arbitrary function

Hash-consing

- Sharing technique for recursive data-structure
  - Every constructed term is stored in *identity* hash table
  - Every construction of term goes through hash table
    - Reuse existing term whenever found
    - Structurally equal terms are physically shared

- Effective caching of operations

Cycle Elimination in Context-sensitive Analysis

- $\mathcal{X}_1 \supseteq_{\rho_1} \mathcal{X}_2 \supseteq_{\rho_2} \cdots \supseteq \mathcal{X}_n \supseteq_{\rho_n} \mathcal{X}_1$
  - Form cycle only for $\rho_1 \sqcap \rho_2 \sqcap \cdots \sqcap \rho_n$
- Eliminate only $\top$-cycles
  - $\mathcal{X}_1 \supseteq_\top \mathcal{X}_2 \supseteq_\top \cdots \supseteq \mathcal{X}_n \supseteq_\top \mathcal{X}_1$
- Analysis is tuned to find $\mathcal{X} \supseteq_\top \mathcal{Y}$ as much as possible
  - Especially with context-insensitive (CI) variables
- If one of $\mathcal{X}_i$ is CI, all $\mathcal{X}_i$ become CI
  - More opportunities for cycle elimination

Experimental Results

Program	Line	HC-CE	HC-NoCE	BDD-CE	BDD-NoCE
make	25K	0.2	0.8	0.5	1.3
m4	35K	1.6	16.4	1.7	16.7
gawk	50K	1.0	3.9	1.2	4.3
tar	56K	1.8	17.2	2.3	18.0
vortex	63K	0.7	1.5	9.8	30.6
sqlite	65K	36.5	156.9	116.4	302.1
gap	78K	1.9	10.6	2.2	4.6
povray	82K	11.3	52.0	37.3	104.0
perlbmk	90K	4.9	40.2	5.3	41.3
bash	127K	12.3	52.3	75.6	183.5
cfengine	129K	24.0	58.0	52.7	188.0
sshd	137K	959.0	11884.0	1090.0	13400.1
python	164K	413.5	1458.6	2555.0	4207.2
vim	224K	20.6	102.0	24.8	111.4
gcc	228K	3.1	4.1	112.9	68.1