Context Sensitive Pointer Analysis with Hash-Consed Forest

Woongsik Choi

KAIST

ROSAEC 2nd Workshop

2009. 7. 9~11

Context Sensitive Pointer Analysis

```
int a,b,c;

void main()
{
  f(&a)¹;

  f(&b)²;

  f(&a)³;
}
```

`void f(int *x)`
`{`
  `g(x,&c)`^{4}`;`
`}`

$[x=\{a\}]^{1}\ \ [x=\{b\}]^{2}\ \ [x=\{a\}]^{3}$

$\longleftrightarrow [x=\{a^{1},b^{2},a^{3}\}]$

$\longleftrightarrow [x=\{a^{\{1,3\}},b^{2}\}]$

`void g(int *y, int *z)`
`{`
  `return;`
`}`

$[y=\{a^{4\bullet\{1,3\}},b^{4\bullet 2}\}\ \ z=\{c^{4}\}]$

- Forest for context representation
- Sometimes partial context is enough

Call Forest

- Program is context free
  - Values defined directly from program are context free

    `g(x,&c)`^{4}`;`
  - Top context for context free values
  - Every context end with top

- First child, next sibling forest

$$
\begin{array}{llll}
\ell & \in & \mathit{Label} & \text{call site label (integer)} \\
\rho & ::= & \top & \text{every possible context} \\
 & | & \bot & \text{no context} \\
 & | & (\ell \cdot \rho^c)|\rho^s & \text{root } \ell,\ \text{ first child } \rho^c,\ \text{ next sibling } \rho^s \\
 & & & \text{siblings are sorted on ascending order on root labels}
\end{array}
$$

Hash-Consing

- Traditional technique from Lisp
  - Share all structurally equal values
  - Through *identity map* implemented as hash table

```
type t = Nil | Cons of int*t
let x = Cons(2,Cons(1,Nil))
let y = Cons(3,Cons(1,Nil))
```

⬇

```
type t = Nil | Cons of int*t*id
let hcons (hd,tl) =
  try  Hash.find table (hd,getid(tl))
  with Notfound ->
    let new = Cons(hd,tl,newid()) in
    Hash.add table (hd,getid(tl)) new;
    new
let x = hcons(2,hcons(1,Nil))
let y = hcons(3,hcons(1,Nil))
```

Maximal Sharing

- Canonical representation
  - Exactly one representation for semantically equal values
- Hash-consing + canonical representation
  - Maximal sharing
  - All semantically equal values are shared
- **`x=[1,2,3]  y=[2,1,3]`**
  - **`x`** and **`y`** are not shared by hash-consing
  - For list semantics, it is maximal sharing
  - For set semantics, it is not maximal sharing
- Canonical representation for set semantics
  - Simple solution : sorted list
  - Sorted siblings in call forest
  - Benefit from maximal sharing must be larger than sorting overhead

Canonical Representation for Top Context

- Many representations for semantically top context
  1. $\top$
  2. Fully explicit context where $\top$ is used only for main
  3. 2 with some subtrees replaced by $\top$
- Reduce all top contexts to $\top$
  1. Before analysis, put one level unfolded top context into hash table
     e.g.) function called at 1,2,3 : $(1 \cdot \top)|(2 \cdot \top)|(3 \cdot \top)|\bot$
  2. Save it for each function
  3. During analysis, compare hash lookup result with saved top.
     If equal, return $\top$ instead
- Call forest vs. Binary Decision Diagram (BDD)
  - BDD is hash-consing on bit level decision tree
  - Call forest can be useful if entire analysis is not represented as BDD