Instance-wise Points-to Analysis for Loop-based Dependence Testing

Peng Wu¹, Paul Feautrier², David Padua³, Zehra Sura³

IBM TJ Watson Research Center¹
INRIA, France²
University of Illinois, Urbana-Champaign³
What is Pointer Analysis?

- **Two flavors of Pointer Analysis**
  - Relations between pointers
    - *Alias* analysis, *shape* analysis
  - Relations between pointers and memory
    - *Points-to* analysis

```
unalias(a,b), unalias(a,c), alias(b,c)
```

```
Points-to(a)={grey}
Points-to(b,c)=Points-to={red}
```
Points-to Analysis for Dependence Test

```java
for (i=0; i<n; i++) {
    a = new ();
    b = new ();
    s: ... = a.x;
    t = a;
    a = b;
    b = t;
    t:b.x = ...;
}
```

(i) at statement `s`

(ii) at statement `t`
Points-to Analysis for Dependence Test

for (i=0; i<n; i++) {
    a = new ();
    b = new ();
    s: ... = a.x;
    t = a;
    a = b;
    b = t;
    t:b.x = ...
}

(i) at statement s

(ii) at statement t
Points-to Analysis for Dependence Test

for (i=0; i<n; i++){
    a = new ();
    b = new ();
    s:  ... = a.x;
    t = a;
    a = b;
    b = t;
    t:b.x = ...;
}
Arrays of Pointers

Another source of pointers

- Java multidimensional arrays
- Dynamic arrays in C/C++
ICS 2002

Instance-wise Points-to Analysis for Loop-based Dependence Testing
Representing Points-to Information

\[
\begin{align*}
0: & \quad a = \text{new Complex}[n]; \\
& \quad \text{for } (i=0; i\leq n; i++)
\begin{align*}
1: & \quad a[i] = \text{new Complex}(); \\
& \quad \text{after statement 1, iteration } i
\end{align*}
\end{align*}
\]

\begin{align*}
\text{after statement 1, iteration } i \\
a & \rightarrow \{N_0\} \\
a[x] & \rightarrow \{\text{null, } (N_1[x], x \leq i)\}
\end{align*}
Unusual aspects of the representation

- Heap objects named by allocation statement instances
  - runtime variables appear in heap names
- Summarize points-to sets of multiple pointers as a mapping
  - Mapping from array indices to heap names
- Conditions are imposed on heap names
Overview of the Algorithm

- An iterative data flow analysis
- Handles Java pointer semantics but not pointer arithmetic
- Context-sensitive, flow-sensitive, and inter-procedural
An Intuitive Flavor of the Analysis

1: Complex[] a = new Complex[n];
   for(i = 0; i ≤ n; i++){
2:     a[i] = new Complex();
   }

4:        a[x]→{null}

1⁺ a[x]→{null}

2⁺ a[x]→{null,(N₂[i], x=i)}
   →{null,(N₂[x], x=i)}

2⁻ a[x]→{null,(N₂[x], x=i⁻)}

2⁺ a[x]→{null,(N₂[x], x=i⁻), (N₂[x], x=i)}
   →{null,(N₂[x], x={i⁻, i})}

2⁻ a[x]→{null,(N₂[x], x=i⁻)}

4 a[x]→{null,(N₂[x], x≤n)}
Aging of Different Depths

0: q = new obj();
1: for(i = 0; i<n; i++) {
2:   a[i] = q;
3:   q = new Obj();
4: }

Aging depth 0 \( (i \rightarrow i^-) \): \( p[x] \rightarrow N_3[i^-], x \leq i \)
Aging depth 1 \( (i \rightarrow i-1) \): \( p[x] \rightarrow N_3[x-1], x_1 \leq i \)
Transfer Function for $a[i] = q$

$$r[x][y] = \begin{cases} 
q \text{ if } \text{addr}(r[x][y]) = \text{addr}(a[i]) \\
r[x][y] \text{ otherwise}
\end{cases}$$

$$r[x] = a \land y = i$$

$$r[x][y] = r[x][y] \cup q, r[x] \cap a \neq \emptyset \land \{\text{null}\} \land y = i$$
Application of Points-to Information

- Dependence test
  - traditional loop transformations (loop interchanging, blocking, parallelization, vectorization)

- Heap analysis
  - null pointer, array bounds checking
Experiment and Evaluation

- 8 numerical Java programs using multidimensional arrays
- Points-to analysis implemented in javac
- Uses Omega library for dependence test after obtaining points-to information
- Applied to dependence testing and null-pointer exception elimination
## Analysis Cost

*prog. points*: # of program points considered  
*ours/javac*: pointer analysis time/javac time

<table>
<thead>
<tr>
<th>Program</th>
<th>Prog. Points</th>
<th>Analysis Time</th>
<th>Line (inline)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>time (ms)</td>
<td>ours/javac</td>
</tr>
<tr>
<td>listtable</td>
<td>10</td>
<td>9</td>
<td>0.1%</td>
</tr>
<tr>
<td>cmatmul</td>
<td>25</td>
<td>162</td>
<td>2.1%</td>
</tr>
<tr>
<td>cshallow</td>
<td>73</td>
<td>259</td>
<td>3.4%</td>
</tr>
<tr>
<td>cholesky</td>
<td>19</td>
<td>195</td>
<td>2.6%</td>
</tr>
<tr>
<td>sor</td>
<td>18</td>
<td>184</td>
<td>2.5%</td>
</tr>
<tr>
<td>lufact</td>
<td>40</td>
<td>168</td>
<td>2.9%</td>
</tr>
<tr>
<td>moldyn</td>
<td>53</td>
<td>77</td>
<td>1.0%</td>
</tr>
<tr>
<td>euler</td>
<td>299</td>
<td>2440</td>
<td>25%</td>
</tr>
</tbody>
</table>
# Dependence Test and Parallelization

**type**: type-based pointer analysis + Omega  
**flat**: flat array + Omega  
**ours**: our pointer analysis + Omega

<table>
<thead>
<tr>
<th>Program</th>
<th># of Dependences</th>
<th># of Parallel Loops</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>type</td>
<td>ours</td>
</tr>
<tr>
<td>listtable</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>cmatmul</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>cholesky</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>shallow</td>
<td>1092</td>
<td>152</td>
</tr>
<tr>
<td>sor</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>lufact</td>
<td>72</td>
<td>45</td>
</tr>
<tr>
<td>moldyn</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>euler</td>
<td>12559</td>
<td>2489</td>
</tr>
</tbody>
</table>

Lower is better  
Higher is better

ICS 2002  
Instance-wise Points-to Analysis for Loop-based Dependence Testing
Summary

A unified algorithm to capture both instance-wise and element-wise points-to information
- Finer-grained heap naming scheme
- Richer points-to mapping representation
- Complete coverage of Java pointer semantics

- Reasonable cost for regular array programs, but still more expensive than traditional pointer analysis
- Points-to information applied to loop-based dependence analysis and array exception elimination
ICS 2002 Instance-wise Points-to Analysis for Loop-based Dependence Testing

Pointer Analysis Road Map

- **Element-wise**:
  - alias analysis
  - alias(a, b) at s
  - a at s vs. b at t

- **Instance-wise**:
  - a[x] at (s, i) vs. b[y] at (t, j)
  - a at (s, i) vs. b at (t, j)
  - a at s vs. b at t

- **Shape analysis**:
  - alias(a.ptr, a.ptr.ptr) at s

- **Points-to analysis**:
  - alias analysis