The Ant and The Grasshopper: Fast and Accurate Pointer Analysis for Millions of Lines of Code

Ben Hardekopf and Calvin Lin
PLDI 2007

(slides put together by Saheel)
Contributions of the Paper

• Identify current state-of-the-art
  - Compare 3 well-known algorithms
  - Fastest takes ½ hour to analyze 1M lines of C

• Advance the state-of-the-art
  - Two techniques for inclusion-based analysis
  - Over 3x faster, same precision
Why Pointer Analysis?

• Pointer information - vital for most program analyses like program verification and program understanding.

• Precise pointer analysis is NP hard.

• The most precise analyses are flow sensitive and context sensitive, but do not scale to large programs.
Example

Flow-sensitive algorithm

Andersen’s algorithm

Steensgaard’s algorithm
Dataflow Equations – Flow-Sensitive

\[ x := \&y \]
\[ G' = G \text{ with } pt'(x) \supseteq \{y\} \]

\[ x := y \]
\[ G' = G \text{ with } pt'(x) \supseteq pt(y) \]

\[ x := *y \]
\[ G' = G \text{ with } pt'(x) \supseteq U pt(a) \text{ for all } a \text{ in } pt(y) \]

\[ *x := y \]
\[ G' = G \text{ with } pt'(a) \cup pt(y) \text{ for all } a \text{ in } pt(x) \]

**strong updates**

**weak update**
Dataflow Equations – Flow-Insensitive

Statements

\[ x := &y \]
\[ G = G \text{ with } \text{pt}(x) \cup \{y\} \]

\[ x := y \]
\[ G = G \text{ with } \text{pt}(x) \cup \text{pt}(y) \]

\[ x := \ast y \]
\[ G = G \text{ with } \text{pt}(x) \cup \text{pt}(a) \text{ for all } a \in \text{pt}(y) \]

\[ \ast x := y \]
\[ G = G \text{ with } \text{pt}(a) \cup \text{pt}(y) \text{ for all } a \in \text{pt}(x) \]

weak updates only
Suppose $S$ and $S_1$ are set-valued variables.

- $S \leftarrow S_1$: strong update = set assignment
- $S \cup \leftarrow S_1$: weak update = set union
  - this is like $S \leftarrow S \cup S_1$
Set Constraints

Statements

**Simple**
- \( x := \& y \)

**Base**
- \( x := \* y \)

**Complex1**
- \( x := y \)

**Complex2**
- \( \* x := y \)
Background: Anderson's algorithm

• Generate constraints from the code
• Build a constraint graph
  • Nodes: variables
  • Edges: inclusion constraints
• Add indirect constraints
  • Pointer dereference
  • Recursively compute transitive closure of the graph
c = &f;
e = &c;
g = &a;
a = d;
b = a;
d = *e;
*e = b;
*e = b;
*g = e;
c = &f;
e = &c;
g = &a;
a = d;
b = a;
d = *e;
*e = b;
*g = e;

c \supseteq \{f\}
e \supseteq \{c\}
g \supseteq \{a\}
a \supseteq d
b \supseteq a
d \supseteq *e
*e \supseteq b
*g \supseteq e
c = &f;
e = &c;
g = &a;
a = d;
b = a;
d = *e;
*e = b;
*g = e;

c ⊇ {f}
e ⊇ {c}
g ⊇ {a}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Background: Inclusion-based Analysis

c = &f;
e = &c;
g = &a;
a = d;
b = a;
d = *e;
*e = b;
*g = e;

c \supseteq \{f\}
e \supseteq \{c\}
g \supseteq \{a\}
a \supseteq d
b \supseteq a
d \supseteq *e
*e \supseteq b
*g \supseteq e
c = &f;
e = &c;
g = &a;
a = d;
b = a;
d = *e;
*e = b;
*g = e;

c ⊇ {f}
e ⊇ {c}
g ⊇ {a}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Background: Inclusion-based Analysis

c ⊇ \{f\}
e ⊇ \{c\}
g ⊇ \{a\}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Background: Inclusion-based Analysis

Constraint Graph

c ⊇ {f}
e ⊇ {c}
g ⊇ {a}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
* g ⊇ e
Background: Inclusion-based Analysis

Constraint Graph

- c ⊇ {f}
- e ⊇ {c}
- g ⊇ {a}
- a ⊇ d
- b ⊇ a
- d ⊇ *e
- *e ⊇ b
- *g ⊇ e
Background: Inclusion-based Analysis

Constraint Graph

c ⊇ {f}
e ⊇ {c}
g ⊇ {a}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Background: Inclusion-based Analysis

Constraint Graph

c ⊇ {f}
e ⊇ {c}
\textcolor{red}{g ⊇ \{a\}}
a ⊇ d
b ⊇ a
d ⊇ e
*e ⊇ b
*\textcolor{red}{g} ⊇ e
Background: Inclusion-based Analysis

Constraint Graph
Background: Inclusion-based Analysis

Constraint Graph

\[
\begin{align*}
  c & \supseteq \{f\} \\
  e & \supseteq \{c\} \\
  g & \supseteq \{a\} \\
  a & \supseteq d \\
  b & \supseteq a \\
  d & \supseteq *e \\
  *e & \supseteq b \\
  *g & \supseteq e
\end{align*}
\]
Background: Inclusion-based Analysis

Constraint Graph

- c ⊇ {f}
- e ⊇ {c}
- g ⊇ {a}
- a ⊇ d
- b ⊇ a
- d ⊇ *e
- *e ⊇ b
- *g ⊇ e
Background: Inclusion-based Analysis

Constraint Graph

- c \supseteq \{f\}
- e \supseteq \{c\}
- g \supseteq \{a\}
- a \supseteq d
- b \supseteq a
- d \supseteq *e
- *e \supseteq b
- *g \supseteq e
Background: Inclusion-based Analysis

Constraint Graph

c ⊇ \{f\}
e ⊇ \{c\}
g ⊇ \{a\}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Background: Inclusion-based Analysis

Constraint Graph

- c \supseteq \{f\}
- e \supseteq \{c\}
- g \supseteq \{a\}
- a \supseteq d
- b \supseteq a
- d \supseteq *e
- *e \supseteq b
- *g \supseteq e
Background: Inclusion-based Analysis

Constraint Graph

- $c \supseteq \{f\}$
- $e \supseteq \{c\}$
- $g \supseteq \{a\}$
- $a \supseteq d$
- $b \supseteq a$
- $d \supseteq *e$
- $*e \supseteq b$
- $*g \supseteq e$
Background: Inclusion-based Analysis

Constraint Graph

- \( c \supseteq \{f\} \)
- \( e \supseteq \{c\} \)
- \( g \supseteq \{a\} \)
- \( a \supseteq d \)
- \( b \supseteq a \)
- \( d \supseteq *e \)
- \( *e \supseteq b \)
- \( *g \supseteq e \)
Background: Inclusion-based Analysis

Constraint Graph
Background: Inclusion-based Analysis

Constraint Graph

Edges:
- c ⊇ {f}
- e ⊇ {c}
- g ⊇ {a}
- a ⊇ d
- b ⊇ a
- d ⊇ *e
- *e ⊇ b
- *g ⊇ e
Background: Inclusion-based Analysis

Constraint Graph

c ⊇ \{f\}
e ⊇ \{c\}
g ⊇ \{a\}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Background: Inclusion-based Analysis

Constraint Graph

\[ \begin{align*}
  c & \supseteq \{f\} \\
  e & \supseteq \{c\} \\
  g & \supseteq \{a\} \\
  a & \supseteq d \\
  b & \supseteq a \\
  d & \supseteq *e \\
  *e & \supseteq b \\
  *g & \supseteq e
\end{align*} \]
Background: Inclusion-based Analysis

Constraint Graph
Background: Inclusion-based Analysis

Constraint Graph

c ⊇ {f}
e ⊇ {c}
g ⊇ {a}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Background: Inclusion-based Analysis

Constraint Graph

$\mathbf{a}$

$\mathbf{f}$

$\mathbf{b}$

$\mathbf{f}$

$\mathbf{c}$

$\mathbf{f}$

$\mathbf{d}$

$\mathbf{f}$

$\mathbf{e}$

$\mathbf{f}$

$\mathbf{g}$

$\mathbf{a}$

$\mathbf{f}$

$\mathbf{c} \supset \{f\}$

$\mathbf{e} \supset \{c\}$

$\mathbf{g} \supset \{a\}$

$\mathbf{a} \supset \mathbf{d}$

$\mathbf{b} \supset \mathbf{a}$

$\mathbf{d} \supset \ast \mathbf{e}$

$\ast \mathbf{e} \supset \mathbf{b}$

$\ast \mathbf{g} \supset \mathbf{e}$
Background: Inclusion-based Analysis

Constraint Graph
Background: Inclusion-based Analysis

Constraint Graph

- c ⊇ {f}
- e ⊇ {c}
- g ⊇ {a}
- a ⊇ d
- b ⊇ a
- d ⊇ *e
- *e ⊇ b
- *g ⊇ e
Background: Inclusion-based Analysis

Constraint Graph

\[
\begin{align*}
    &c \supseteq \{f\} \\
    &e \supseteq \{c\} \\
    &g \supseteq \{a\} \\
    &a \supseteq d \\
    &b \supseteq a \\
    &d \supseteq *e \\
    &*e \supseteq b \\
    &*g \supseteq e
\end{align*}
\]
Background: Inclusion-based Analysis

Constraint Graph

c ⊆ \{f\}
e ⊆ \{c\}
g ⊆ \{a\}
a ⊆ d
b ⊆ a
d ⊆ *e
*e ⊆ b
*g ⊆ e
Background: Inclusion-based Analysis

Constraint Graph

c \supseteq \{f\}
e \supseteq \{c\}
g \supseteq \{a\}
a \supseteq d
b \supseteq a
d \supseteq *e
*e \supseteq b
*\#g \supseteq e
Background: Inclusion-based Analysis

Constraint Graph

- **c ⊇ \{f\}**
- **e ⊇ \{c\}**
- **g ⊇ \{a\}**
- **a ⊇ d**
- **b ⊇ a**
- **d ⊇ *e**
- ***e ⊇ b**
- ***g ⊇ e**
Background: Inclusion-based Analysis

Constraint Graph
• Inclusion-based analysis is $O(n^3)$

• Optimize with **online cycle detection**
  - All nodes in the same cycle will have identical points-to sets
  - Most cycles appear *during* the analysis as new edges are added
Cycle detection mechanism will largely determine performance of analysis.

Must carefully balance aggression versus overhead:
- Too aggressive → too much graph traversal
- Too conservative → cycles found too late
Contributions

• Two new techniques for cycle detection
  – Lazy Cycle Detection
  – Hybrid Cycle Detection

• Techniques are complementary
  – Hybrid Cycle Detection can be composed with any other cycle detection technique
1. Lazy Cycle Detection

• Well-known fact:
  A cycle forces nodes to have identical points-to sets

• Key Insight:
  Nodes with identical points-to sets indicate possible cycles
  - Balance aggression and overhead by waiting for the effect of the cycle (identical points-to sets) to become obvious
Lazy Cycle Detection - Example

Constraint Graph

c ⊇ \{f\}
e ⊇ \{c\}
g ⊇ \{a\}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Lazy Cycle Detection

Constraint Graph

- Constraint: $c \supset \{f\}$
- Constraint: $e \supset \{c\}$
- Constraint: $g \supset \{a\}$
- Constraint: $a \supset d$
- Constraint: $b \supset a$
- Constraint: $d \supset *e$
- Constraint: $*e \supset b$
- Constraint: $*g \supset e$
Lazy Cycle Detection

Constraint Graph

c \supseteq \{f\}
e \supseteq \{c\}
g \supseteq \{a\}
a \supseteq d
b \supseteq a
d \supseteq *e
*e \supseteq b
*g \supseteq e
Lazy Cycle Detection

Constraint Graph

c ⊇ \{f\}
e ⊇ \{c\}
g ⊇ \{a\}
a ⊇ d
b ⊇ a
d ⊇ *e
* e ⊇ b
* g ⊇ e
Lazy Cycle Detection

Constraint Graph
Lazy Cycle Detection

Constraint Graph

c ⊇ {f}
e ⊇ {c}
g ⊇ {a}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Lazy Cycle Detection

Constraint Graph

- $c \supset \{f\}$
- $e \supset \{c\}$
- $g \supset \{a\}$
- $a \supset d$
- $b \supset a$
- $d \supset *e$
- $*e \supset b$
- $*g \supset e$
Lazy Cycle Detection

Constraint Graph
Lazy Cycle Detection

Constraint Graph

c ⊆ \{f\}
e ⊆ \{c\}
g ⊆ \{a\}
a ⊆ d
b ⊆ a
d ⊆ *e
*e ⊆ b
*g ⊆ e
Lazy Cycle Detection

Constraint Graph
Lazy Cycle Detection

Constraint Graph
Lazy Cycle Detection

Constraint Graph
Lazy Cycle Detection

Constraint Graph

c ⊆ \{f\}
e ⊆ \{c\}
g ⊆ \{a\}
a ⊆ d
b ⊆ a
d ⊆ *e
*e ⊆ b
*g ⊆ e
Lazy Cycle Detection

Constraint Graph

\[
\begin{align*}
    c & \supseteq \{f\} \\
    e & \supseteq \{c\} \\
    g & \supseteq \{a\} \\
    a & \supseteq d \\
    b & \supseteq a \\
    d & \supseteq \ast e \\
    \ast e & \supseteq b \\
    \ast g & \supseteq e
\end{align*}
\]
Lazy Cycle Detection

Constraint Graph

c ⊇ {f}
e ⊇ {c}
g ⊇ {a}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Lazy Cycle Detection

Constraint Graph
Lazy Cycle Detection

Constraint Graph

c \supseteq \{f\}
e \supseteq \{c\}
g \supseteq \{a\}
a \supseteq d
b \supseteq a
d \supseteq *e
*e \supseteq b
*\text{g} \supseteq e
Lazy Cycle Detection

Constraint Graph

c ⊇ \{f\}
e ⊇ \{c\}
g ⊇ \{a\}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Lazy Cycle Detection

Constraint Graph
Lazy Cycle Detection

Constraint Graph

- c ⊇ {f}
- e ⊇ {c}
- g ⊇ {a}
- a ⊇ d
- b ⊇ a
- d ⊇ *e
- *e ⊇ b
- *g ⊇ e
Lazy Cycle Detection

• **IS LAZY** because cycles are detected only while propagating constraints to them (well after they are created)

• Nodes with identical points-to sets MAY NOT be part of a cycle

• An Additional heuristic is needed to stop this wasteful search: Don’t trigger cycles detection on the same edge twice

• => Cycle detection is not guaranteed to find all cycles
Contributions

• Two new techniques for cycle detection
  – Lazy Cycle Detection
  – Hybrid Cycle Detection

• Techniques are complementary
  – Hybrid Cycle Detection can be composed with any other cycle detection technique
Hybrid Cycle Detection

<table>
<thead>
<tr>
<th>cheap</th>
<th>finds few cycles</th>
<th>finds many cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline Cycle Detection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expensive</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hybrid Cycle Detection

<table>
<thead>
<tr>
<th>cheap</th>
<th>finds few cycles</th>
<th>finds many cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline Cycle Detection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>expensive</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Online Cycle Detection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Hybrid Cycle Detection

**Key Insight:** combining offline and online techniques can give us the best of both worlds.

PS: Offline = before actual constraint graph traversal

<table>
<thead>
<tr>
<th></th>
<th>finds few cycles</th>
<th>finds many cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>cheap</td>
<td><strong>Offline Cycle Detection</strong></td>
<td><strong>Hybrid Cycle Detection</strong></td>
</tr>
<tr>
<td>expensive</td>
<td><strong>Online Cycle Detection</strong></td>
<td></td>
</tr>
</tbody>
</table>
Hybrid Cycle Detection

- Eagerly finds cycles *without* traversing constraint graph
- Not guaranteed to find all cycles – 46–74% in these benchmarks
- Can be combined with other cycle detection techniques
Hybrid Cycle Detection

• Offline component

• Online component
Hybrid Cycle Detection – Offline

c ⊇ \{f\}
e ⊇ \{c\}
g ⊇ \{a\}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Ignore Base Constraints!

c \supseteq \{f\}
e \supseteq \{c\}
g \supseteq \{a\}
a \supseteq d
b \supseteq a
d \supseteq *e
*e \supseteq b
*g \supseteq e
Hybrid Cycle Detection – Offline

c ⊃ {f}
e ⊃ {c}
g ⊃ {a}
a ⊃ d
b ⊃ a
d ⊃ *e
*e ⊃ b
*g ⊃ e
Hybrid Cycle Detection – Offline

Offline Constraint Graph
Hybrid Cycle Detection – Offline

Offline Constraint Graph

c \supseteq \{f\}
e \supseteq \{c\}
g \supseteq \{a\}
a \supseteq d
b \supseteq a
d \supseteq *e
*e \supseteq b
*g \supseteq e
Hybrid Cycle Detection – Offline

Offline Constraint Graph
Hybrid Cycle Detection – Offline

Offline Constraint Graph

c ⊇ {f}
e ⊇ {c}
g ⊇ {a}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Hybrid Cycle Detection – Offline

Offline Constraint Graph
Hybrid Cycle Detection – Offline

Offline Constraint Graph

\[ e \rightarrow \{a, b, d\} \]
Hybrid Cycle Detection

- Offline component
- Online component
Hybrid Cycle Detection – Online

Constraint Graph

**e** → \{a, b, d\}

e ⊆ \{c\}
g ⊆ \{a\}
a ⊆ d
b ⊆ a
d ⊆ *e
*e ⊆ b
*g ⊆ e
Hybrid Cycle Detection – Online

Constraint Graph
Hybrid Cycle Detection – Online

Constraint Graph

Hybrid Cycle Detection – Online

Constraint Graph
Hybrid Cycle Detection – Online

Constraint Graph

e → \{a, b, d\}

Constraints:
- \(e \supset \{c\}\)
- \(g \supset \{a\}\)
- \(a \supset d\)
- \(b \supset a\)
- \(d \supset *e\)
- \(*e \supset b\)
- \(*g \supset e\)
Hybrid Cycle Detection – Online

Constraint Graph

- e
- c
- g
- a
- f

a/b/c/d

e → {a, b, d}

e ⊇ {c}
g ⊇ {a}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Hybrid Cycle Detection – Online

Constraint Graph

- **e** → {a, b, d}
- e ⊇ {c}
- g ⊇ {a}
- a ⊇ d
- b ⊇ a
- d ⊇ *e
- *e ⊇ b
- *g ⊇ e
Hybrid Cycle Detection – Online

Constraint Graph
Hybrid Cycle Detection – Online

Constraint Graph

e → \{a, b, d\}

Constraint:
- e ⊇ \{c\}
- g ⊇ \{a\}
- a ⊇ d
- b ⊇ a
- d ⊇ *e
- *e ⊇ b
- *g ⊇ e
Hybrid Cycle Detection – Online

Constraint Graph

e → \{a,b,d\}

e ⊇ \{c\}
g ⊇ \{a\}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Hybrid Cycle Detection – Online

Constraint Graph

e → \{a, b, d\}

e ⊇ \{c\}
g ⊇ \{a\}
a ⊇ d
b ⊇ a
d ⊇ *e
*e ⊇ b
*g ⊇ e
Hybrid Cycle Detection – Online

Constraint Graph
Hybrid Cycle Detection – Online

Constraint Graph

\[ e \rightarrow \{a, b, d\} \]

\[ e \supset \{c\} \]
\[ g \supset \{a\} \]
\[ a \supset d \]
\[ b \supset a \]
\[ d \supset *e \]
\[ *e \supset b \]
\[ *g \supset e \]
Hybrid Cycle Detection – Online

Constraint Graph

e → \{a,b,d\}

e \supseteq \{c\}
g \supseteq \{a\}
a \supseteq d
b \supseteq a
d \supseteq *e
*e \supseteq b
*g \supseteq e
Hybrid Cycle Detection – Online

Constraint Graph

e → \{a, b, d\}

e \supseteq \{c\}
g \supseteq \{a\}
a \supseteq d
b \supseteq a
d \supseteq *e
*e \supseteq b
*\ g \supseteq e
Hybrid Cycle Detection – Online

Constraint Graph

e → \{a, b, d\}

\begin{align*}
\text{e} & \supset \{c\} \\
\text{g} & \supset \{a\} \\
\text{a} & \supset \text{d} \\
\text{b} & \supset \text{a} \\
\text{d} & \supset ^*\text{e} \\
^*\text{e} & \supset \text{b} \\
^*\text{g} & \supset \text{e}
\end{align*}
Evaluation

Compare against 3 well-known algorithms
• Heintze and Tardieu [PLDI'01]
• Pearce et al [PASTE'04]
• Berndl et al [PLDI'03]

All algorithms compute the exact same solution

Six benchmarks: 100K—2M LOC
• Emacs, Ghostscript, Gimp, Insight, Wine, Linux Kernel
### Dataset

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Original Constraints</th>
<th>Reduced Constraints</th>
<th>Base</th>
<th>Simple</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emacs-21.4a</td>
<td>169K</td>
<td>83,213</td>
<td>21,460</td>
<td>4,088</td>
<td>11,095</td>
<td>6,277</td>
</tr>
<tr>
<td>Ghostscript-8.15</td>
<td>242K</td>
<td>169,312</td>
<td>67,310</td>
<td>12,154</td>
<td>25,880</td>
<td>29,276</td>
</tr>
<tr>
<td>Gimp-2.2.8</td>
<td>554K</td>
<td>411,783</td>
<td>96,483</td>
<td>17,083</td>
<td>43,878</td>
<td>35,522</td>
</tr>
<tr>
<td>Insight-6.5</td>
<td>603K</td>
<td>243,404</td>
<td>85,375</td>
<td>13,198</td>
<td>35,382</td>
<td>36,795</td>
</tr>
<tr>
<td>Wine-0.9.21</td>
<td>1,338K</td>
<td>713,065</td>
<td>171,237</td>
<td>39,166</td>
<td>62,499</td>
<td>69,572</td>
</tr>
<tr>
<td>Linux-2.4.26</td>
<td>2,172K</td>
<td>574,788</td>
<td>203,733</td>
<td>25,678</td>
<td>77,936</td>
<td>100,119</td>
</tr>
</tbody>
</table>

Constraints reduced using offline variable substitution (60-77%)
Figure 6. Performance (in seconds) of our new combined algorithm (LCD+HCD) versus three state-of-the-art inclusion-based algorithms. Note that the Y-axis is log-scale.
## Results – details...

<table>
<thead>
<tr>
<th></th>
<th>Emacs</th>
<th>Ghostscript</th>
<th>Gimp</th>
<th>Insight</th>
<th>Wine</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCD-Offline</td>
<td>0.05</td>
<td>0.17</td>
<td>0.26</td>
<td>0.23</td>
<td>0.51</td>
<td>0.62</td>
</tr>
<tr>
<td>HT</td>
<td>1.66</td>
<td>12.03</td>
<td>59.00</td>
<td>42.49</td>
<td>1,388.51</td>
<td>393.30</td>
</tr>
<tr>
<td>PKH</td>
<td>2.05</td>
<td>20.05</td>
<td>92.30</td>
<td>117.88</td>
<td>1,946.16</td>
<td>1,181.59</td>
</tr>
<tr>
<td>BLQ</td>
<td>4.74</td>
<td>121.60</td>
<td>167.56</td>
<td>265.94</td>
<td>5,117.64</td>
<td>5,144.29</td>
</tr>
<tr>
<td>LCD</td>
<td>3.07</td>
<td>15.23</td>
<td>39.50</td>
<td>39.02</td>
<td>1,157.10</td>
<td>327.65</td>
</tr>
<tr>
<td>HCD</td>
<td>0.46</td>
<td>49.55</td>
<td>59.70</td>
<td>73.92</td>
<td><strong>OOM</strong></td>
<td>659.74</td>
</tr>
<tr>
<td>HT+HCD</td>
<td>0.46</td>
<td>7.29</td>
<td>11.94</td>
<td>14.82</td>
<td>643.89</td>
<td>102.77</td>
</tr>
<tr>
<td>PKH+HCD</td>
<td>0.46</td>
<td>10.52</td>
<td>17.12</td>
<td>21.91</td>
<td>838.08</td>
<td>114.45</td>
</tr>
<tr>
<td>BLQ+HCD</td>
<td>5.81</td>
<td>115.00</td>
<td>173.46</td>
<td>257.05</td>
<td>4,211.71</td>
<td>4,581.91</td>
</tr>
<tr>
<td>LCD+HCD</td>
<td>0.56</td>
<td>7.99</td>
<td>12.50</td>
<td>15.97</td>
<td>492.40</td>
<td>86.74</td>
</tr>
</tbody>
</table>

**Table 3.** Performance (in seconds), using bitmaps for points-to-sets. The HCD-Offline analysis is reported separately and not included in the times for those algorithms using HCD. The HCD algorithm runs out of memory on the Wine benchmark.
## Results – Memory Consumption

<table>
<thead>
<tr>
<th></th>
<th>Emacs</th>
<th>Ghostscript</th>
<th>Gimp</th>
<th>Insight</th>
<th>Wine</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>HT</td>
<td>17.7</td>
<td>84.9</td>
<td>279.0</td>
<td>231.5</td>
<td>1,867.2</td>
<td>901.3</td>
</tr>
<tr>
<td>PKH</td>
<td>17.6</td>
<td>83.9</td>
<td>269.5</td>
<td>194.7</td>
<td>1,448.3</td>
<td>840.7</td>
</tr>
<tr>
<td>BLQ</td>
<td>215.6</td>
<td>216.1</td>
<td>216.2</td>
<td>216.1</td>
<td>216.2</td>
<td>216.2</td>
</tr>
<tr>
<td>LCD</td>
<td>14.3</td>
<td>74.6</td>
<td>269.0</td>
<td>184.4</td>
<td>1,465.1</td>
<td>830.1</td>
</tr>
<tr>
<td>HCD</td>
<td>18.1</td>
<td>138.7</td>
<td>416.1</td>
<td>290.5</td>
<td><strong>OOM</strong></td>
<td>1,301.5</td>
</tr>
<tr>
<td>HT+HCD</td>
<td>12.4</td>
<td>80.8</td>
<td>253.9</td>
<td>186.5</td>
<td>1,391.4</td>
<td>842.5</td>
</tr>
<tr>
<td>PKH+HCD</td>
<td>13.9</td>
<td>79.1</td>
<td>264.6</td>
<td>186.0</td>
<td>1,430.2</td>
<td>807.5</td>
</tr>
<tr>
<td>BLQ+HCD</td>
<td>215.8</td>
<td>216.2</td>
<td>216.2</td>
<td>216.2</td>
<td>216.2</td>
<td>216.2</td>
</tr>
<tr>
<td>LCD+HCD</td>
<td>13.9</td>
<td>73.5</td>
<td>263.9</td>
<td>183.6</td>
<td>1,406.4</td>
<td>807.9</td>
</tr>
</tbody>
</table>
Concluding Remarks

• 20x faster than Berndl et al, 6x faster than Pearce et al and 3x faster than Heintze et al

• No precision lost!

• Anderson's algorithm made scalable to millions of nodes
Thank you!
Backup: Transitive Closure Algorithm

let $G = \langle V, E \rangle$

$W \leftarrow V$

while $W \neq \emptyset$ do

$n \leftarrow \text{SELECT-FROM}(W)$

for each $v \in \text{pts}(n)$ do

for each constraint $a \supseteq \ast n$ do

if $v \rightarrow a \not\in E$ then

$E \leftarrow E \cup \{v \rightarrow a\}$

$W \leftarrow W \cup \{v\}$

for each constraint $\ast n \supseteq b$ do

if $b \rightarrow v \not\in E$ then

$E \leftarrow E \cup \{b \rightarrow v\}$

$W \leftarrow W \cup \{b\}$

for each $n \rightarrow z \in E$ do

$\text{pts}(z) \leftarrow \text{pts}(z) \cup \text{pts}(n)$

if $\text{pts}(z)$ changed then

$W \leftarrow W \cup \{z\}$
Lazy Cycle Detection Algorithm

```
let $G = < V, E >$
$R \leftarrow \emptyset$
$W \leftarrow V$
while $W \neq \emptyset$ do
    $n \leftarrow \text{SELECT-FROM}(W)$
    for each $v \in \text{pts}(n)$ do
        for each constraint $a \supseteq *n$ do
            if $v \rightarrow a \notin E$ then
                $E \leftarrow E \cup \{v \rightarrow a\}$
                $W \leftarrow W \cup \{v\}$
        for each constraint $*n \supseteq b$ do
            if $b \rightarrow v \notin E$ then
                $E \leftarrow E \cup \{b \rightarrow v\}$
                $W \leftarrow W \cup \{b\}$
    for each $n \rightarrow z \in E$ do
        if $\text{pts}(z) = \text{pts}(n) \land n \rightarrow z \notin R$ then
            $\text{DETECT-AND-COLLAPSE-CYCLES}(z)$
            $R \leftarrow R \cup \{n \rightarrow z\}$
            $\text{pts}(z) \leftarrow \text{pts}(z) \cup \text{pts}(n)$
            if $\text{pts}(z)$ changed then
                $W \leftarrow W \cup \{z\}$
```
Heintze and Tardieu [PLDI'01]

• Online algorithm
• During construction, as new inclusion edges are added to the graph, the transitive edges are NOT added
• During analysis, indirect constraints are resolved through REACHABILITY queries.
• => lots of redundant queries

• In their paper, they reported results for field based implementation. When fields are expanded, the algorithm is dramatically slow.
Pearce et al [PASTE'04]

- Two variants

- First:
  - Maintain topological order of the graph
  - A newly inserted edge that violates this ordering COULD create a cycle, so check whenever this happens.

- Second
  - Periodic sweep of the constraint graph to detect and collapse cycles.
Field sensitive inclusion-based pointer analysis for JAVA programs

Uses BDDs to represent graph and points-to sets

This paper extends this algorithm by
  • Making it field insensitive
  • Handle indirect function calls