Analyzing Million Lines of C: 
A General Sparse Analysis Framework

Kwangkeun Yi

Programming Research Laboratory
Seoul National University

04/23/2012 @ CSAIL MIT

Co-work with Hakjoo Oh, Kihong Heo, Wonchan Lee, Wooseok Lee
[PLDI’12, VMCAI’11, APLAS’11, …]
Static Analysis: Sound, Unsound, Useful

Automatic sound estimation of sw behaviors before execution
- under many names
  - theory
  - pl, se, veri.
  - cmplr

  "abstract interpretation"
  "type system", "model checking", "theorem proving"
  "data-flow analysis", etc.

sound & precise analysis

sound analysis

unsound analysis
Dichotomy: “bug-finders” vs. “verifiers”

- Scalable bug-finders
- Unscalable verifiers
Dichotomy: “bug-finders” vs. “verifiers”

- Scalable: bug-finders
- Unscalable: verifiers

- Unsound
- Sound

Monday, April 23, 2012
Dichotomy: “bug-finders” vs. “verifiers”

- scalable
  - bug-finders
- unsound
- sound
- unsound

Our contribution?
Our Story

- In 2007, we commercialized
  - sound in design, unsound yet scalable in reality
  - memory-­bug-­finding tool for full C, non domain-­specific
- Realistic workbench available
  - “let’s try to scale-­up its sound & global version”
Our Scalability Improvement

Sparrow
The Early Bird

sound-&-global version

- < 1.4M in 10hrs with intervals
- < 0.14M in 20hrs with octagons
How we achieved
- sound design of Sparrow
- spatial & temporal localizations

Sparse analysis framework
- general for AI-based analyzers for C-like languages
- precision-preserving

“An important strength is that the theoretical result is very general. It could be applied to many other analyses. PLDI papers have been accepted that were simply instances of this framework.” (from PLDI reviews)
Static Analysis Example: Abstract Equations

\[ x = \text{readInt}; \]
1:
\[ \text{while } (x \leq 99) \]
2:
\[ x++; \]
3:
\[ \text{end} \]
4:

Capture the dynamics by abstract equations; solve; reason.

\[
\begin{align*}
    x_1 &= [\infty, +\infty] \text{ or } x_3 \\
    x_2 &= x_1 \text{ and } [\infty, 99] \\
    x_3 &= x_2 + 1 \\
    x_4 &= x_1 \text{ and } [100, +\infty]
\end{align*}
\]
How to Design Sound Static Analyses?

Abstract Interpretation [CousotCousot]: a powerful design theory

- How to derive correct yet arbitrarily precise equations?
  - Non-obvious: ptrs, heap, exns, high-order ftns, etc.

\[
\begin{align*}
x &= \text{readInt;} \\
\text{while } (x \leq 99) &\quad \Rightarrow \quad \text{how?} \\
\quad &\quad \Rightarrow \quad x_1 = [-\infty, +\infty] \text{ or } x_3 \\
x_2 &= x_1 \text{ and } [-\infty, 99] \\
x_3 &= x_2 + 1 \\
x_4 &= x_1 \text{ and } [100, +\infty]
\end{align*}
\]

- Define an abstract semantics function \( \hat{F} \) s.t. \( \cdots \)

- How to solve the equations in a finite time?

\[
\begin{align*}
x_1 &= [-\infty, +\infty] \text{ or } x_3 \\
x_2 &= x_1 \text{ and } [-\infty, 99] \\
x_3 &= x_2 + 1 \\
x_4 &= x_1 \text{ and } [100, +\infty]
\end{align*}
\]

- Fixpoint iterations for an upperbound of \( \text{fix} \hat{F} \)
Design of Sparrow
• Designed in the *abstract interpretation* framework
• To find memory safety violations in C
  • buffer-overrun, memory leak, null deref., etc.
  • flow-sensitive values analysis for int & ptrs (static + dynamic)
• for the full set of C
Program

\[ \langle \mathbb{C}, \rightarrow \rangle \]

- \( \mathbb{C} \) : set of program points
- \( \rightarrow \subseteq \mathbb{C} \times \mathbb{C} \) : control flow relation

\[ c' \rightarrow c \quad (c \text{ is the next command to } c') \]

Commands

\[ lv := e \mid lv := \text{alloc}(a) \mid \text{assume}(x < e) \mid \text{call}(f_x, e) \mid \text{return}_f \]

expression \( e \rightarrow n \mid e + e \mid lv \mid \&lv \)

l-value \( lv \rightarrow x \mid *e \mid e[e] \mid e.x \)

allocation \( a \rightarrow [e]_l \mid \{x\}_l \)
Abstract Semantics

- One abstract state $\in \hat{S}$ that subsumes all reachable states at each program point

\[
[\hat{P}] \in C \rightarrow \hat{S} = \text{fix} \hat{F} \\
\hat{S} = \hat{L} \rightarrow \hat{V}
\]

- Abstract semantic function

\[
\hat{F} \in (C \rightarrow \hat{S}) \rightarrow (C \rightarrow \hat{S}) \\
\hat{F}(\hat{X}) = \lambda c \in C. \hat{f}_c(\bigcup_{c' \rightarrow c} \hat{X}(c'))
\]

\[
\hat{f}_c \in \hat{S} \rightarrow \hat{S} : \text{abstract semantics at point } c
\]

\[
\hat{L} = \text{Var} + \text{AllocSite} + \text{AllocSite} \times \text{FieldName} \\
\hat{V} = \hat{Z} \times 2^\hat{L} \times 2^\text{AllocSite} \times \hat{Z} \times 2^\text{AllocSite} \times 2^\text{FieldName} \\
\hat{Z} = \{[l, u] \mid l, u \in \mathbb{Z} \cup \{-\infty, +\infty\} \land l \leq u \} \cup \{\perp\}
\]
Computing $\text{fix } F = \bigsqcup_{i \in \mathbb{N}} F^i(\bot)$

\[
\hat{F}(\hat{X}) = \lambda c \in \mathbb{C}. f_c(\bigsqcup_{c' \leftarrow c} \hat{X}(c')).
\]

| $W \in \text{Worklist} = 2^\mathbb{C}$ |
| $\hat{X} \in \mathbb{C} \rightarrow \hat{\mathbb{S}}$ |
| $f_c \in \hat{\mathbb{S}} \rightarrow \hat{\mathbb{S}}$ |
| $W := \mathbb{C}$ |
| $\hat{X} := \lambda c. \bot$ |
| repeat |
| $c := \text{choose}(W)$ |
| $\hat{s} := \hat{f}_c(\bigsqcup_{c' \leftarrow c} X(c'))$ |
| if $\hat{s} \not\subseteq \hat{X}(c)$ |
| $W := W \cup \{c' \in \mathbb{C} | c \leftarrow c'\}$ |
| $\hat{X}(c) := \hat{X}(c) \sqcup \hat{s}$ |
| until $W = \emptyset$ |

Naive fixpoint algorithm

Worklist algorithm
The Algorithms Too Weak To Scale

less-382 (23,822 LoC)

Monday, April 23, 2012
Improving Scalability

Key Idea: Localization

“Right Part at Right Moment”
Spatial & Temporal Localizations

\[
x = x + 1
\]

\[
y = y - 1
\]

\[
z = x
\]

\[
v = y
\]

\[
\text{ret } *a + *b
\]
Spatial & Temporal Localizations
Spatial & Temporal Localizations

\[
x = x + 1
\]

\[
y = y - 1
\]

\[
z = x
\]

\[
v = y
\]

\[
\text{ret } *a + *b
\]
x = x + 1
y = y - 1
z = x
v = y
ret *a + *b
Spatial & Temporal Localizations

\[
x = x + 1
\]

\[
y = y - 1
\]

\[
z = x
\]

\[
v = y
\]

\[
\text{ret } *a + *b
\]
Spatial & Temporal Localizations

\[ x = x + 1 \]

\[ y = y - 1 \]

\[ z = x \]

\[ v = y \]

\[ \text{ret } \ast a + \ast b \]
Spatial & Temporal Localizations

\[ x = x + 1 \]
\[ y = y - 1 \]
\[ z = x \]
\[ v = y \]
\[ \text{ret } ^a + ^b \]

Monday, April 23, 2012
Spatial & Temporal Localizations

\[
\begin{align*}
x &= x + 1 \\
y &= y - 1 \\
z &= x \\
v &= y \\
\text{ret } &\ast a + \ast b
\end{align*}
\]
x = x + 1
y = y - 1
z = x
v = y
ret *a + *b
Spatial & Temporal Localizations

\[ x = x + 1 \]

\[ y = y - 1 \]

\[ z = x \]

\[ v = y \]

\[ \text{ret } *a + *b \]
Spatial & Temporal Localizations

- \( x = x + 1 \)
- \( y = y - 1 \)
- \( z = x \)
- \( v = y \)
- \( \text{ret } *a + *b \)
Spatial & Temporal Localizations

\[
x = x + 1
\]

\[
y = y - 1
\]

\[
z = x
\]

\[
v = y
\]

\[
\text{ret } *a + *b
\]
Spatial & Temporal Localizations

\[
x = x + 1
\]
\[
y = y - 1
\]
\[
z = x
\]
\[
v = y
\]
\[
\text{ret } *a + *b
\]
Spatial & Temporal Localizations

\[ x = x + 1 \]
\[ y = y - 1 \]
\[ z = x \]
\[ v = y \]
\[ \text{ret } *a + *b \]
Spatial & Temporal Localizations

x = x + 1

y = y - 1

z = x

v = y

ret *a + *b
Spatial & Temporal Localizations

x = x + 1

y = y - 1

z = x

v = y

ret *a + *b
Spatial & Temporal Localizations

x = x + 1

y = y - 1

z = x

v = y

ret *a + *b
Spatial & Temporal Localizations

\[
\begin{align*}
  x &= x + 1 \\
  y &= y - 1 \\
  z &= x \\
  v &= y \\
  \text{ret} &= \ast a + \ast b
\end{align*}
\]
\[ \hat{X}, \hat{X}' \in \mathbb{C} \rightarrow \mathbb{S} \]
\[ \hat{f}_c \in \mathbb{S} \rightarrow \mathbb{S} \]
\[ \hat{X} := \hat{X}' := \lambda c. \perp \]
repeat
\[ \hat{X}' := \hat{X} \]
for all \( c \in \mathbb{C} \) do
\[ \hat{X}(c) := \hat{f}_c(\hat{X}(c')) \]
until \( \hat{X} \subseteq \hat{X}' \)
Spatial Localization
Spatial Localization
(“framing”, “abstract gc”)

call f

return
Vital in Analysis Practice

On average, 755 re-analysis per procedure
But Existing Approach is Too Conservative

huge room for localizations than reachability-based technique

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>accessed memory / reachable memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>spell-1.0</td>
<td>2,213</td>
<td>5 / 453 (1.1%)</td>
</tr>
<tr>
<td>barcode-0.96</td>
<td>4,460</td>
<td>19 / 1175 (1.6%)</td>
</tr>
<tr>
<td>httptunnel-3.3</td>
<td>6,174</td>
<td>10 / 673 (1.5%)</td>
</tr>
<tr>
<td>gzip-1.2.4a</td>
<td>7,327</td>
<td>22 / 1002 (2.2%)</td>
</tr>
<tr>
<td>jwhois-3.0.1</td>
<td>9,344</td>
<td>28 / 830 (3.4%)</td>
</tr>
<tr>
<td>parser</td>
<td>10,900</td>
<td>75 / 1787 (4.2%)</td>
</tr>
<tr>
<td>bc-1.06</td>
<td>13,093</td>
<td>24 / 824 (2.9%)</td>
</tr>
<tr>
<td>less-290</td>
<td>18,449</td>
<td>86 / 1546 (5.6%)</td>
</tr>
</tbody>
</table>

average: only 4%
Hurdle: Accessed Locations Before Analysis?

- Yes, by yet another analysis
- The pre-analysis must be quick
- The pre-analysis must be safe
  - over-estimating the accessed abstract locs
Our Pre-analysis

For Safely Estimating the Accessed Abstract Locations

- one further abstraction
- correct design
  \[ C \rightarrow \hat{S} \xleftarrow{\gamma} \hat{\hat{S}} \]
- abstract semantic function: flow-insensitive
  \[ \hat{F}_p = \lambda \hat{s}. \left( \bigsqcup_{c \in C} \hat{f}_c(\hat{s}) \right) \]
Implement in the Abstract Semantics for Call Cmd

\[ \hat{f}_c \in \hat{S} \rightarrow \hat{S} \]

\[
\hat{f}_c(\hat{s}) = \begin{cases} 
\hat{s}[\hat{\mathcal{L}}(lv)(\hat{s}) \xrightarrow{w} \hat{\mathcal{V}}(e)(\hat{s})] & \text{cmd}(c) = lv := e \\
\hat{s}[\hat{\mathcal{L}}(lv)(\hat{s}) \xrightarrow{w} \langle \bot, \bot, \{\langle l, [0, 0], \hat{\mathcal{V}}(e)(\hat{s}).1 \rangle, \bot \rangle] & \text{cmd}(c) = lv := \text{alloc}([e]_l) \\
\hat{s}[\hat{\mathcal{L}}(lv)(\hat{s}) \xrightarrow{w} \langle \bot, \bot, \bot, \{\langle l, \{x\} \rangle \} \rangle] & \text{cmd}(c) = lv := \text{alloc}({x}_l) \\
\hat{s}[x \mapsto \langle \hat{s}(x).1 \cap [-\infty, u(\hat{\mathcal{V}}(e)(\hat{s}).1)]\rangle, \hat{s}(x).2, \hat{s}(x).3, \hat{s}(x).4] & \text{cmd}(c) = \text{assume}(x < e) \\
\hat{s}[x \mapsto \hat{\mathcal{V}}(e)(\hat{s})]|\text{access}(f) & \text{cmd}(c) = \text{call}(f_x, e) \\
\hat{s} & \text{cmd}(c) = \text{return}_f 
\end{cases}
\]

\[ \text{access}(f) = \bigcup_{g \in \text{callee}(f)} \left( \bigcup_{c \in \text{control}(g)} \mathcal{A}(c)(\hat{s}_{\text{pre}}) \right) \]
Performance of sound & global Sparrow

<table>
<thead>
<tr>
<th>Programs</th>
<th>LOC</th>
<th>Interval\texttt{vanilla}</th>
<th>Interval\texttt{base}</th>
<th>Spd↑</th>
<th>Mem↓</th>
<th>Interval\texttt{sparse}</th>
<th>Dep</th>
<th>Fix</th>
<th>Total</th>
<th>Mem</th>
<th>D(c)</th>
<th>U(c)</th>
<th>Spd↑</th>
<th>Mem↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip-1.2.4a</td>
<td>7K</td>
<td>772</td>
<td>14</td>
<td>65</td>
<td>55 x</td>
<td>73 %</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>63</td>
<td>2.4</td>
<td>2.5</td>
<td>5 x</td>
<td>3 %</td>
</tr>
<tr>
<td>bc-1.06</td>
<td>13K</td>
<td>1,270</td>
<td>96</td>
<td>126</td>
<td>13 x</td>
<td>54 %</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>75</td>
<td>4.6</td>
<td>4.9</td>
<td>14 x</td>
<td>40 %</td>
</tr>
<tr>
<td>tar-1.13</td>
<td>20K</td>
<td>12,947</td>
<td>338</td>
<td>177</td>
<td>38 x</td>
<td>80 %</td>
<td>6</td>
<td>2</td>
<td>8</td>
<td>93</td>
<td>2.9</td>
<td>2.9</td>
<td>42 x</td>
<td>47 %</td>
</tr>
<tr>
<td>less-382</td>
<td>23K</td>
<td>9,561</td>
<td>1,113</td>
<td>1,211</td>
<td>8 x</td>
<td>66 %</td>
<td>27</td>
<td>6</td>
<td>33</td>
<td>127</td>
<td>11.9</td>
<td>11.9</td>
<td>37 x</td>
<td>66 %</td>
</tr>
<tr>
<td>make-3.76.1</td>
<td>27K</td>
<td>24,240</td>
<td>1,391</td>
<td>1,893</td>
<td>13 x</td>
<td>68 %</td>
<td>16</td>
<td>5</td>
<td>21</td>
<td>114</td>
<td>5.8</td>
<td>5.8</td>
<td>90 x</td>
<td>74 %</td>
</tr>
<tr>
<td>wget-1.9</td>
<td>35K</td>
<td>44,092</td>
<td>1,214</td>
<td>378</td>
<td>36 x</td>
<td>85 %</td>
<td>8</td>
<td>3</td>
<td>11</td>
<td>85</td>
<td>2.4</td>
<td>2.4</td>
<td>110 x</td>
<td>78 %</td>
</tr>
<tr>
<td>screen-4.0.2</td>
<td>45K</td>
<td>∞</td>
<td>31,324</td>
<td>3,996</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>767</td>
<td>303</td>
<td>53.0</td>
<td>54.0</td>
<td>41 x</td>
<td>92 %</td>
</tr>
<tr>
<td>a2ps-4.14</td>
<td>64K</td>
<td>N/A</td>
<td>3,200</td>
<td>1,392</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>40</td>
<td>353</td>
<td>2.6</td>
<td>2.8</td>
<td>80 x</td>
<td>75 %</td>
</tr>
<tr>
<td>bash-2.05a</td>
<td>105K</td>
<td>N/A</td>
<td>1,683</td>
<td>1,386</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>67</td>
<td>220</td>
<td>3.0</td>
<td>3.0</td>
<td>25 x</td>
<td>84 %</td>
</tr>
<tr>
<td>lsh-2.0.4</td>
<td>111K</td>
<td>N/A</td>
<td>45,522</td>
<td>5,266</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>471</td>
<td>577</td>
<td>21.1</td>
<td>21.2</td>
<td>97 x</td>
<td>89 %</td>
</tr>
<tr>
<td>sendmail-8.13.6</td>
<td>130K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>517</td>
<td>744</td>
<td>20.7</td>
<td>20.7</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>nethack-3.3.0</td>
<td>211K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>517</td>
<td>227</td>
<td>4.5</td>
<td>4.5</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>vim60</td>
<td>227K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>227</td>
<td>744</td>
<td>20.7</td>
<td>20.7</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>emacs-22.1</td>
<td>399K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>14,126</td>
<td>16,373</td>
<td>5,298</td>
<td>72.4</td>
<td>72.4</td>
<td>N/A</td>
</tr>
<tr>
<td>python-2.5.1</td>
<td>435K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>14,126</td>
<td>16,373</td>
<td>5,298</td>
<td>72.4</td>
<td>72.4</td>
<td>N/A</td>
</tr>
<tr>
<td>linux-3.0</td>
<td>710K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>14,126</td>
<td>16,373</td>
<td>5,298</td>
<td>72.4</td>
<td>72.4</td>
<td>N/A</td>
</tr>
<tr>
<td>gimp-2.6</td>
<td>959K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>227</td>
<td>744</td>
<td>20.7</td>
<td>20.7</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ghostscript-9.00</td>
<td>1,363K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>14,126</td>
<td>16,373</td>
<td>5,298</td>
<td>72.4</td>
<td>72.4</td>
<td>N/A</td>
</tr>
</tbody>
</table>

- **Loc**: Lines of code
- **Time**: Analysis time in seconds
- **Mem**: Memory consumption in megabytes
- **Spd↑**: Speedup over baseline
- **Mem↓**: Memory saving

Thanks to sparse analysis technique, the performance numbers are 62%–592% faster than non-relational analysis. The average size of abstract locations increases by 54x–562% over top of the in-place packing strategy for all analyzers. Though our general-purpose packing heuristic is similar to Min’s, our packing is more practical and scales to 574 KLO% within 69 mins and 2:4 G. We also compared main analysis time and peak memory consumption of the various versions of analyses.

We discuss the relation between performance and sparsity in average. Because our abstract locations are defined in real programs, only a few abstract locations are defined and used in relational analysis.

The average sizes of variable packs per program point are defined. For example, the average size of variable packs of emacs is 74 times bigger than the one of ghostscript. Even though ghostscript is a large difference of sparsity/average size between emacs and ghostscript, the analysis performance is more dependent on the sparsity than abstract domain and semantics. In exactly the same engineering efforts, we replaced interval-based abstract domain by octagon-based domain. Octagon forms the access-based localization in terms of variable packs. The three analyzers use the same way as interval analysis. The performance numbers are saves a lot of memory, but the analysis is still not scale.

Variable Packing

- ** Dep**: Data dependency analysis steps
- ** Fix**: Actual analysis steps
- ** Total**: Total analysis steps
- ** Mem**: Memory consumption

The average size of variable packs per program point is 627–592% over top of the in-place packing strategy for all analyzers. Though our general-purpose packing heuristic is similar to Min’s, our packing is more practical and scales to 574 KLO% within 69 mins and 2:4 G. The average size of abstract locations is defined in real programs. It clearly shows the key observation to sparse analysis in this sparsity of analysis. We could achieve orders of magnitude speed up compared to the baseline possible. Thanks to sparse analysis technique, the performance numbers are 62%–592% faster than non-relational analysis. The average size of abstract locations increases by 54x–562% over top of the in-place packing strategy for all analyzers. Though our general-purpose packing heuristic is similar to Min’s, our packing is more practical and scales to 574 KLO% within 69 mins and 2:4 G.
Temporal Localization
(and spatial localization automatically follows)
Temporal Localization

- Don’t blindly follow the control flow of pgm text
- Follow the dependency of statement semantics
- from definition points directly to their use points

\[ \hat{X}, \hat{X}' \in C \rightarrow \hat{S} \]
\[ f_c \in \hat{S} \rightarrow \hat{S} \]
\[ \hat{X} := \hat{X}' := \lambda c. \perp \]
repeat
\[ \hat{X}' := \hat{X} \]
for all \( c \in C \) do
\[ \hat{X}(c) := \hat{f}_c( \bigcup_{c' \leftarrow c} \hat{X}(c')) \]
until \( \hat{X} \subseteq \hat{X}' \)
Temporal Localization

• Don’t blindly follow the control flow of pgm text
• Follow the dependency of statement semantics
• from definition points directly to their use points

\[
\hat{X}, \hat{X}' \in C \rightarrow \hat{S} \\
\hat{f}_c \in \hat{S} \rightarrow \hat{S} \\
\hat{X} := \hat{X}' := \lambda c. \bot \\
\text{repeat} \\
\quad \hat{X}' := \hat{X} \\
\quad \text{for all } c \in C \text{ do} \\
\quad \quad \hat{X}(c) := \hat{f}_c(\hat{\rho}_{c' \leftarrow c}(\hat{X}(c'))) \\
\text{until } \hat{X} \subseteq \hat{X}'
\]
Temporal Localization

\[ x = x + 1 \]
\[ y = y - 1 \]
\[ z = x \]

vs.

\[ x = x + 1 \]
\[ y = y - 1 \]
\[ z = x \]
Precision Preserving Sparse Analysis Framework

\[ \hat{F} : \hat{D} \rightarrow \hat{D} \quad \xrightarrow{\text{sparsify}} \quad \hat{F}_s : \hat{D} \rightarrow \hat{D} \]

\[ \text{fix } \hat{F} \quad \overset{\text{still}}{=} \quad \text{fix } \hat{F}_s \]
Towards Sparse Version

Analyzer computes the fixpoint of \( \hat{F} \in (C \rightarrow \hat{S}) \rightarrow (C \rightarrow \hat{S}) \)

- baseline non-sparse one
  \[ \hat{F}(\hat{X}) = \lambda c \in C. f_c( \bigsqcup_{c' \sim c} \hat{X}(c')) \]

- unrealizable sparse version
  \[ \hat{F}_s(\hat{X}) = \lambda c \in C. f_c( \bigsqcup_{c' \sim l} \hat{X}(c')|_l) \]

- realizable sparse version
  \[ \hat{F}_a(\hat{X}) = \lambda c \in C. f_c( \bigsqcup_{c' \sim a} \hat{X}(c')|_l) \]
Unrealizable Sparse One

\[ \hat{F}_s(\hat{X}) = \lambda c \in \mathbb{C}. \hat{f}_c( \bigcup \hat{X}(c')|l) \]

Data Dependency

\[ c_0 \xrightarrow{l} c_n \triangleq \exists c_0 \ldots c_n \in \text{Paths}, l \in \hat{L}. \]
\[ l \in D(c_0) \cap U(c_n) \land \forall i \in (0, n). l \notin D(c_i) \]
Unrealizable Sparse One

\[ \hat{f}_s(\hat{X}) = \lambda c \in \mathbb{C}. \hat{f}_c( \bigsqcup_{c' \sim c} \hat{X}(c') | l). \]

Data Dependency

\[ c_0 \xrightarrow{l} c_n \triangleq \exists c_0 \ldots c_n \in \text{Paths}, l \in \hat{L}. \]
\[ l \in D(c_0) \cap U(c_n) \land \forall i \in (0, n). l \notin D(c_i) \]

Def-Use Sets

\[ D(c) \triangleq \{ l \in \hat{L} | \exists \hat{s} \subseteq \bigsqcup_{c' \sim c} S(c'). \hat{f}_c(\hat{s})(l) \neq \hat{s}(l) \} \]

\[ U(c) \triangleq \{ l \in \hat{L} | \exists \hat{s} \subseteq \bigsqcup_{c' \sim c} S(c'). \hat{f}_c(\hat{s})|_{D(c)} \neq \hat{f}_c(\hat{s} \setminus l)|_{D(c)} \} \]
Unrealizable Sparse One

\[
\hat{F}_s(\hat{X}) = \lambda c \in \mathbb{C}. \hat{f}_c( \bigsqcup \hat{X}(c')|_l).
\]

Data Dependency

\[
c_0 \xrightarrow{l} c_n \ \overset{\Delta}{=} \ \exists c_0 \ldots c_n \in \text{Paths}, l \in \hat{L}.

l \in D(c_0) \cap U(c_n) \land \forall i \in (0, n). l \notin D(c_i)
\]

Def-Use Sets

\[
D(c) \overset{\Delta}{=} \{ l \in \hat{L} \mid \exists \hat{s} \subseteq \bigsqcup_{c' \rightarrow c} S(c'). \hat{f}_c(\hat{s})(l) \neq \hat{s}(l) \}
\]

\[
U(c) \overset{\Delta}{=} \{ l \in \hat{L} \mid \exists \hat{s} \subseteq \bigsqcup_{c' \rightarrow c} S(c'). \hat{f}_c(\hat{s})|_{D(c)} \neq \hat{f}_c(\hat{s}\setminus l)|_{D(c)} \}
\]

Precision Preserving

\[
\text{fix } \hat{F} = \text{fix } \hat{F}_s \quad \text{modulo } D
\]
Data Dependency Example

\[
x := &y \quad *p := &z \quad y := x
\]

Def \{x\} \quad \{v, w\} \quad \{y\}

Use \emptyset \quad \{p, v, w\} \quad \{x\}
Realizable Sparse One

\[ \hat{F}_a(\hat{X}) = \lambda c \in C. f_c(\bigcup_{c' \sim_a c} \hat{X}(c')|l). \]

Realizable Data Dependency

\[ c_0 \overset{l}{\sim}_a c_n \overset{\Delta}{=} \exists c_0 \ldots c_n \in \text{Paths}, l \in \hat{L}. \]

\[ l \in \hat{D}(c_0) \cap \hat{U}(c_n) \land \forall i \in (0, n).l \notin \hat{D}(c_i) \]
Realizable Sparse One

\( \hat{F}_a(\hat{X}) = \lambda c \in \mathbb{C}. \hat{f}_c( \bigcup_{c' \sim l, a} \hat{X}(c')) |_l \).

Realizable Data Dependency

\( c_0 \sim_l^a c_n \triangleq \exists c_0 \ldots c_n \in \text{Paths}, l \in \mathbb{L}. \)

\( l \in \hat{D}(c_0) \cap \hat{U}(c_n) \land \forall i \in (0, n). l \notin \hat{D}(c_i) \)

Precision Preserving

\( \text{fix } \hat{F} = \text{fix } \hat{F}_a \mod \hat{D} \)

If the following conditions hold
Conditions on $\hat{D} \& \hat{U}$

- over-approximation
  \[ \hat{D}(c) \supseteq D(c) \land \hat{U}(c) \supseteq U(c) \]

- spurious definitions should be also included in uses
  \[ \hat{D}(c) - D(c) \subseteq \hat{U}(c) \]
Why the Conditions of $\hat{D}$ & $\hat{U}$

- $x := &y$
- $*p := &z$
- $y := x$

**Def**
- $\{x\}$
- $\{v, w\}$
- $\{y\}$

**Use**
- $\emptyset$
- $\{p, v, w\}$
- $\{x\}$
Why the Conditions of $\hat{D} \& \hat{U}$

\[\begin{align*}
\text{Def} & \quad \{x\} & \{v, w, x\} & \{y\} \\
\text{Use} & \quad \emptyset & \{p, v, w\} & \{x\}
\end{align*}\]
Why the Conditions of $\hat{D} \& \hat{U}$

\[
\begin{align*}
&\text{Def} \quad \{x\} \quad \{v, w, x\} \quad \{y\} \\
&\text{Use} \quad \emptyset \quad \{p, v, w\} \quad \{x\}
\end{align*}
\]
Why the Conditions of $\hat{D} \& \hat{U}$

\[ \begin{align*}
\text{Def} & \quad \{x\} & \{v, w, x\} & \{y\} \\
\text{Use} & \quad \emptyset & \{p, v, w\} & \{x\}
\end{align*} \]
Why the Conditions of $\hat{D} \& \hat{U}$

\[
x := &y \\
*p := &z \\
y := x
\]

**Def**
- $\{x\}$
- $\{v, w, x\}$
- $\{y\}$

**Use**
- $\emptyset$
- $\{p, v, w, x\}$
- $\{x\}$
Why the Conditions of $\hat{D}$ & $\hat{U}$

\[
\begin{align*}
\text{Def} & \quad \{x\} & \{v, w, x\} & \{y\} \\
\text{Use} & \quad \emptyset & \{p, v, w, x\} & \{x\}
\end{align*}
\]
Hurdle: ∗∗∗ & ∗∗∗ Before Analysis?

- Yes, by yet another analysis with further abstraction
- correct design
  \[ \mathbb{C} \to \hat{\mathbb{S}} \xrightarrow{\gamma} \hat{\mathbb{S}} \]
- abstract semantic function: flow-insensitive
  \[ \hat{F}_p = \lambda \hat{s}. (\bigsqcup_{c \in \mathbb{C}} \hat{f}_c(\hat{s})) \]
Performance of sound & global Sparrow

Table 3: Performance numbers are limited within each of syntactic % blocks. We also group abstract program uses the same way as interval analysis. Semantic functions are appropriately changed, except ab.

---

### Programs

<table>
<thead>
<tr>
<th>Programs</th>
<th>LOC</th>
<th>Interval&lt;sub&gt;vanilla&lt;/sub&gt;</th>
<th>Interval&lt;sub&gt;base&lt;/sub&gt;</th>
<th>Mem↓&lt;sub&gt;1&lt;/sub&gt;</th>
<th>Interval&lt;sub&gt;sparse&lt;/sub&gt;</th>
<th>Spd↑</th>
<th>Mem↓&lt;sub&gt;2&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip-1.2.4a</td>
<td>7K</td>
<td>772</td>
<td>240</td>
<td>14</td>
<td>65</td>
<td>55 x</td>
<td>73 %</td>
</tr>
<tr>
<td>bc-1.06</td>
<td>13K</td>
<td>1,270</td>
<td>276</td>
<td>96</td>
<td>126</td>
<td>13 x</td>
<td>54 %</td>
</tr>
<tr>
<td>tar-1.13</td>
<td>20K</td>
<td>12,947</td>
<td>881</td>
<td>338</td>
<td>177</td>
<td>38 x</td>
<td>80 %</td>
</tr>
<tr>
<td>less-382</td>
<td>23K</td>
<td>9,561</td>
<td>1,113</td>
<td>1,211</td>
<td>378</td>
<td>8 x</td>
<td>66 %</td>
</tr>
<tr>
<td>make-3.76.1</td>
<td>27K</td>
<td>24,240</td>
<td>1,391</td>
<td>1,893</td>
<td>443</td>
<td>13 x</td>
<td>68 %</td>
</tr>
<tr>
<td>wget-1.9</td>
<td>35K</td>
<td>44,092</td>
<td>2,546</td>
<td>1,214</td>
<td>378</td>
<td>36 x</td>
<td>85 %</td>
</tr>
<tr>
<td>screen-4.0.2</td>
<td>45K</td>
<td>∞</td>
<td>N/A</td>
<td>31,324</td>
<td>3,996</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>a2ps-4.14</td>
<td>64K</td>
<td>∞</td>
<td>N/A</td>
<td>3,200</td>
<td>1,392</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>bash-2.05a</td>
<td>105K</td>
<td>∞</td>
<td>N/A</td>
<td>1,683</td>
<td>1,386</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>lsh-2.0.4</td>
<td>111K</td>
<td>∞</td>
<td>N/A</td>
<td>45,522</td>
<td>5,266</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>sendmail-8.13.6</td>
<td>130K</td>
<td>∞</td>
<td>N/A</td>
<td>517</td>
<td>227</td>
<td>744</td>
<td>678</td>
</tr>
<tr>
<td>nethack-3.3.0</td>
<td>211K</td>
<td>∞</td>
<td>N/A</td>
<td>14,126</td>
<td>2,427</td>
<td>16,373</td>
<td>5,298</td>
</tr>
<tr>
<td>vim60</td>
<td>227K</td>
<td>∞</td>
<td>N/A</td>
<td>17,518</td>
<td>6,280</td>
<td>23,798</td>
<td>5,190</td>
</tr>
<tr>
<td>emacs-22.1</td>
<td>399K</td>
<td>∞</td>
<td>N/A</td>
<td>29,552</td>
<td>8,278</td>
<td>37,830</td>
<td>7,795</td>
</tr>
<tr>
<td>python-2.5.1</td>
<td>435K</td>
<td>∞</td>
<td>N/A</td>
<td>9,677</td>
<td>1,362</td>
<td>11,039</td>
<td>5,535</td>
</tr>
<tr>
<td>linux-3.0</td>
<td>710K</td>
<td>∞</td>
<td>N/A</td>
<td>26,669</td>
<td>6,949</td>
<td>33,618</td>
<td>20,529</td>
</tr>
<tr>
<td>gimp-2.6</td>
<td>959K</td>
<td>∞</td>
<td>N/A</td>
<td>3,751</td>
<td>123</td>
<td>3,874</td>
<td>3,602</td>
</tr>
<tr>
<td>ghostscript-9.00</td>
<td>1,363K</td>
<td>∞</td>
<td>N/A</td>
<td>14,116</td>
<td>698</td>
<td>14,814</td>
<td>6,384</td>
</tr>
</tbody>
</table>

---

**Table 2**: 

<table>
<thead>
<tr>
<th>Programs</th>
<th>LOC</th>
<th>Interval&lt;sub&gt;vanilla&lt;/sub&gt;</th>
<th>Interval&lt;sub&gt;base&lt;/sub&gt;</th>
<th>Mem↓&lt;sub&gt;1&lt;/sub&gt;</th>
<th>Interval&lt;sub&gt;sparse&lt;/sub&gt;</th>
<th>Spd↑</th>
<th>Mem↓&lt;sub&gt;2&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip-1.2.4a</td>
<td>7K</td>
<td>772</td>
<td>240</td>
<td>14</td>
<td>65</td>
<td>55 x</td>
<td>73 %</td>
</tr>
<tr>
<td>bc-1.06</td>
<td>13K</td>
<td>1,270</td>
<td>276</td>
<td>96</td>
<td>126</td>
<td>13 x</td>
<td>54 %</td>
</tr>
<tr>
<td>tar-1.13</td>
<td>20K</td>
<td>12,947</td>
<td>881</td>
<td>338</td>
<td>177</td>
<td>38 x</td>
<td>80 %</td>
</tr>
<tr>
<td>less-382</td>
<td>23K</td>
<td>9,561</td>
<td>1,113</td>
<td>1,211</td>
<td>378</td>
<td>8 x</td>
<td>66 %</td>
</tr>
<tr>
<td>make-3.76.1</td>
<td>27K</td>
<td>24,240</td>
<td>1,391</td>
<td>1,893</td>
<td>443</td>
<td>13 x</td>
<td>68 %</td>
</tr>
<tr>
<td>wget-1.9</td>
<td>35K</td>
<td>44,092</td>
<td>2,546</td>
<td>1,214</td>
<td>378</td>
<td>36 x</td>
<td>85 %</td>
</tr>
<tr>
<td>screen-4.0.2</td>
<td>45K</td>
<td>∞</td>
<td>N/A</td>
<td>31,324</td>
<td>3,996</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>a2ps-4.14</td>
<td>64K</td>
<td>∞</td>
<td>N/A</td>
<td>3,200</td>
<td>1,392</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>bash-2.05a</td>
<td>105K</td>
<td>∞</td>
<td>N/A</td>
<td>1,683</td>
<td>1,386</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>lsh-2.0.4</td>
<td>111K</td>
<td>∞</td>
<td>N/A</td>
<td>45,522</td>
<td>5,266</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>sendmail-8.13.6</td>
<td>130K</td>
<td>∞</td>
<td>N/A</td>
<td>517</td>
<td>227</td>
<td>744</td>
<td>678</td>
</tr>
<tr>
<td>nethack-3.3.0</td>
<td>211K</td>
<td>∞</td>
<td>N/A</td>
<td>14,126</td>
<td>2,427</td>
<td>16,373</td>
<td>5,298</td>
</tr>
<tr>
<td>vim60</td>
<td>227K</td>
<td>∞</td>
<td>N/A</td>
<td>17,518</td>
<td>6,280</td>
<td>23,798</td>
<td>5,190</td>
</tr>
<tr>
<td>emacs-22.1</td>
<td>399K</td>
<td>∞</td>
<td>N/A</td>
<td>29,552</td>
<td>8,278</td>
<td>37,830</td>
<td>7,795</td>
</tr>
<tr>
<td>python-2.5.1</td>
<td>435K</td>
<td>∞</td>
<td>N/A</td>
<td>9,677</td>
<td>1,362</td>
<td>11,039</td>
<td>5,535</td>
</tr>
<tr>
<td>linux-3.0</td>
<td>710K</td>
<td>∞</td>
<td>N/A</td>
<td>26,669</td>
<td>6,949</td>
<td>33,618</td>
<td>20,529</td>
</tr>
<tr>
<td>gimp-2.6</td>
<td>959K</td>
<td>∞</td>
<td>N/A</td>
<td>3,751</td>
<td>123</td>
<td>3,874</td>
<td>3,602</td>
</tr>
<tr>
<td>ghostscript-9.00</td>
<td>1,363K</td>
<td>∞</td>
<td>N/A</td>
<td>14,116</td>
<td>698</td>
<td>14,814</td>
<td>6,384</td>
</tr>
</tbody>
</table>

---

**Table 3**: Programs are considered as an extreme case. Octagon requires extremely large amount of time, the analysis still not scalable over Octagon but does not perform the localizations and saves a lot of memory. The analysis is still not scalable, though our general-purpose packing strategy for all analyzers makes the analysis realistic but does not perform the localizations and saves a lot of memory.
Existing Sparse Techniques
(developed mostly in dfa community)

- Different notion of data dependency

\[ c_0 \xrightarrow{l_{\text{du}}} c_n \triangleq \exists c_0 \ldots c_n \in \text{Paths}, l \in \hat{L}. \]
\[ l \in D(c_0) \cap U(c_n) \land \forall i \in (0, n). l \notin D_{\text{always}}(c_i) \]

- fail to preserve the original accuracy

- Not general for arbitrary analysis for full C

- tightly coupled with particular analysis (e.g. pointer analysis for “simple” subsets of C)
Summing Up
Our Sparse Framework

• Define a global safe abstract interpreter

\[ \hat{F} \in (\mathbb{C} \rightarrow \hat{S}) \rightarrow (\mathbb{C} \rightarrow \hat{S}) \]

\[ \hat{F}(\hat{X}) = \lambda c \in \mathbb{C}. \hat{f}_c( \bigsqcup_{c' \sim_c} \hat{X}(c')) \]  

• Make it sparse (s&t) with our data dependencies
  • by using a safe pre-analysis for safe def/use sets

\[ \hat{F}_a(\hat{X}) = \lambda c \in \mathbb{C}. \hat{f}_c( \bigsqcup_{c' \sim_a c} \hat{X}(c')|_l) \]

• Resulting sparse one scales with the same result
Thank you.

for technical details

ropas.snu.ac.kr/~kwang/paper/12-pldi-ohheleleyi.pdf