Design and Implementation of Sparse Global Analyses for C-like Languages

Wonchan Lee
wclee@ropas.snu.ac.kr
(with Hakjoo Oh, Kihong Heo, Woosuk Lee, and Kwangkeun Yi)

April 3rd, 2012
@Sunblaze group meeting
Dichotomy: “Bug-finders” vs. “Verifiers”

- **Bug-finders**
  - Scalable
  - Sound

- **Verifiers**
  - Not scalable
  - Unsound
Dichotomy: “Bug-finders” vs. “Verifiers”

- **Scalable**
  - **Bug finders**
    - nearly scalable or nearly scalable and many false negatives
    - many false negatives
    - < 10M LOC
  - **Verifiers**
    - nearly scalable or nearly scalable and many false alarms
    - many false alarms
    - < 0.1M LOC

- **Not scalable**
  - unsound
  - sound
Goal: Sound, Precise, and Scalable Analysis

- Scalable
- Not scalable

- Sound
- Unsound

- Bug finders
- Verifiers
- Nirvana
Our Contributions

scalable

not scalable

unsound

sound

- bug finders
- verifiers
- sparse analysis framework
  - general for AI-based analyzers
  - precision-preserving

nirvana
Our Contributions

- Bug finders: scalable, unsound
  - < 1.4M with intervals
- Sparse analyzers: not scalable, sound
  - < 0.14M with octagons
- Verifiers: scalable, unsound
• Source of inefficiency in static analysis
• Sparse analysis
• Experiment results
Reachable states at each control point

```
y = 2
```
```
x = 1
```
```
z = y + 3
```
```
ret 3 * x
```
Reachable states at each control point

<table>
<thead>
<tr>
<th>y</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
</tr>
</tbody>
</table>

y = 2

<table>
<thead>
<tr>
<th>y</th>
<th>2</th>
</tr>
</thead>
</table>

x = 1

<table>
<thead>
<tr>
<th>y</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>5</td>
</tr>
<tr>
<td>ret</td>
<td>3</td>
</tr>
</tbody>
</table>

z = y + 3

<table>
<thead>
<tr>
<th>y</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>5</td>
</tr>
<tr>
<td>ret</td>
<td>3</td>
</tr>
</tbody>
</table>

ret 3 * x

<table>
<thead>
<tr>
<th>y</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>5</td>
</tr>
<tr>
<td>ret</td>
<td>3</td>
</tr>
</tbody>
</table>
Values are used and defined in a sparse manner.

```
x = 1
z = y + 3
ret 3 * x
```
Some values are *unnecessarily propagated* until they're actually used.

```
ret 3 * x
z = y + 3
x = 1
y = 2
```
What Really Happens

\[ x = y \]

\[ y = y + 1 \]

giant loop A that doesn’t touch \( y \)

\[ x = y \]
What Really Happens

\[ y = y + 1 \]

Giant loop A that doesn’t touch \( y \)

\[ x = y \]

Re-compute A whenever \( y \) changes
Idea: propagating definitions directly to their uses

```
y = 2
x = 1
z = y + 3
ret 3 * x
```
Exploiting Sparsity with DFG

Idea: propagating definitions directly to their uses with DFG
Exploiting Sparsity with DFG

Idea: propagating definitions directly to their uses with DFG
Definitions and uses are sometimes determined at runtime - Mostly because of pointers in C-like languages

```
y = 2
*p = 1
z = y + 3
ret 3 * x
```
Definitions and uses are sometimes determined at runtime - Mostly because of pointers in C-like languages

```
  \*p = 1
  z = y + 3
  ret 3 \* x
```

what are defined?

what are used?
Technical Hurdle

Definitions and uses are sometimes determined at runtime - Mostly because of pointers in C-like languages

\[
\begin{align*}
  \ast p &= 1 \\
  z &= y + 3 \\
  \text{ret} &= 3 \ast x
\end{align*}
\]
Case 1) strong update

\[
p = 1 \\
\text{ret } 3 \times x \\
z = y + 3 \\
y = 2 \\
p\{x\} \\
x = 1
\]
Case 1) strong update

- \( p = 1 \)
- \( x = 1 \)
- \( y = 2 \)
- \( z = y + 3 \)
- \( \text{ret } 3 \times x \)
Case 2) weak update

```
p = 1
ret 3 * x
z = y + 3
y = 2

<table>
<thead>
<tr>
<th>p</th>
<th>{x, y}</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
</tr>
</tbody>
</table>

```

The diagram illustrates the execution flow with specific values assigned to variables based on the weak update case.
Case 2) weak update

- \( p = 1 \)
- \( z = y + 3 \)
- \( y = 2 \)
- \( x = 1 \)

Table:

<table>
<thead>
<tr>
<th></th>
<th>( x, y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y )</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>( x )</td>
<td>1</td>
</tr>
</tbody>
</table>

Ret: \( 3 \times x \)
Case 2) weak update

\[ \text{ret } 3 \times x \]

\[ z = y + 3 \]

\[ \ast p = 1 \]

\[ y = 2 \]

\[ p \{x, y\} \]

\[ y \{1, 2\} \]

\[ x \ 1 \]

wait!

isn’t points-to set a priori unknown?
Technical Hurdle

Case 2) weak update

\[
\begin{align*}
\text{ret } & 3 \times x \\
z & = y + 3 \\
p & \{x, y\} \\
y & \{1, 2\} \\
x & 1
\end{align*}
\]

wait!

isn’t points-to set a priori unknown?

⇒ approximation!
Approximation of Definitions and Uses

$c$:

\[ \ast p = 1 \]

\[ z = y + 3 \]

\[ \text{ret } 3 \ast x \]

\[ y = 2 \]

\[ \text{U}(c) \quad \text{uses of primary analysis} \]

\[ \text{D}(c) \quad \text{definitions of primary analysis} \]
Approximation of Definitions and Uses

\[ c : \quad *p = 1 \]

\[ y = 2 \]

\[ z = y + 3 \]

\[ \text{ret } 3 * x \]

\[ U(c) \subseteq \hat{U}(c) \]

\[ D(c) \subseteq \hat{D}(c) \]
\[ *p = 1 \]
\[ z = y + 3 \]
\[ y = 2 \]
\[ \text{safety condition} \]
\[ \hat{D}(c) - D(c) \subseteq \hat{U}(c) \]
\[ U(c) \subseteq \hat{U}(c) \]
\[ D(c) \subseteq \hat{D}(c) \]

: all spurious definitions should be spurious uses
Example of Unsafe Approximation

$D(c) = \{x\}$
$U(c) = \{\}$

most precise DFG when $p = \{x\}$

$\hat{D}(c) = \{x, y\}$
$\hat{U}(c) = \{\}$

DFG with approximated def/use
Example of Unsafe Approximation

\[ D(c) = \{ x \} \]
\[ U(c) = \{ \} \]

\[ \hat{D}(c) = \{ x, y \} \]
\[ \hat{U}(c) = \{ \} \]

most precise DFG when \( p = \{ x \} \)

no value is really defined here!
Example of Safe Approximation

\[ D(c) = \{ x \} \]
\[ U(c) = \{ \} \]

Most precise DFG when \( p = \{ x \} \)

\[ \hat{D}(c) = \{ x, y \} \]
\[ \hat{U}(c) = \{ y \} \]

DFG with approximated def/use

\[ D(c) = \{ x \} \]
\[ U(c) = \{ \} \]

Most precise DFG when \( p = \{ x \} \)
Example of Safe Approximation

\[ D(c) = \{x\} \]
\[ U(c) = \{\} \]

\[ \hat{D}(c) = \{x, y\} \]
\[ \hat{U}(c) = \{y\} \]

- \( y = 2 \)
- \( *p = 1 \)
- \( z = y + 3 \)
- \( \text{ret } 3 * x \)

most precise DFG when \( p = \{x\} \)

still the correct definition of \( y \) is propagated!

DFG with approximated def/use
Sparse Analysis Preserves Precision

Original Static Analysis

\[ \hat{F}(\hat{X}) = \lambda c. \hat{f}_c( \bigsqcup_{c' \leftrightarrow c} \hat{X}(c')) \]

Sparse Static Analysis

\[ \hat{F}_s(\hat{X}) = \lambda c. \hat{f}_c( \bigsqcup_{c' \xrightarrow{x} c} \hat{X}(c')|_x) \]
Sparse Analysis Preserves Precision

Original Static Analysis

\[ \hat{F}(\hat{X}) = \lambda c.\hat{f}_c( \bigcup_{c' \leftarrow c} \hat{X}(c')) \]

Sparse Static Analysis

\[ \hat{F}_s(\hat{X}) = \lambda c.\hat{f}_c( \bigcup_{c' \xrightarrow{x} c} \hat{X}(c')|_x) \]
Sparse Analysis Preserves Precision

Original Static Analysis

\[ \hat{F}(\hat{X}) = \lambda c. \hat{f}_c( \bigcup_{c' \sim c} \hat{X}(c')) \]

Sparse Static Analysis

\[ \hat{F}_s(\hat{X}) = \lambda c. \hat{f}_c( \bigcup_{c' \sim c} \hat{X}(c')|_x) \]

Theorem 1.

\[ \text{lfp}(\hat{F}) \simeq \text{lfp}(\hat{F}_s) \]
Sparse Analysis Preserves Precision

Original Static Analysis

\[ \hat{F}(\hat{X}) = \lambda c. \hat{f}_c( \bigsqcup_{c' \rightarrow c} \hat{X}(c')) \]

Sparse Static Analysis

\[ \hat{F}_s(\hat{X}) = \lambda c. \hat{f}_c( \bigsqcup_{c' \xrightarrow{x} c} \hat{X}(c') |_x) \]

**Theorem 1.** \( \text{lfp}(\hat{F}) \cong \text{lfp}(\hat{F}_s) \)

same precision up to actually used locations
Experiment Results

Benchmarks

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Functions</th>
<th>Statements</th>
<th>Blocks</th>
<th>maxSCC</th>
<th>AbsLocs</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip-1.2.4a</td>
<td>7K</td>
<td>132</td>
<td>6,446</td>
<td>4,152</td>
<td>2</td>
<td>1,784</td>
</tr>
<tr>
<td>bc-1.06</td>
<td>13K</td>
<td>132</td>
<td>10,368</td>
<td>4,731</td>
<td>1</td>
<td>1,619</td>
</tr>
<tr>
<td>tar-1.13</td>
<td>20K</td>
<td>221</td>
<td>12,199</td>
<td>8,586</td>
<td>13</td>
<td>3,245</td>
</tr>
<tr>
<td>less-382</td>
<td>23K</td>
<td>382</td>
<td>23,367</td>
<td>9,207</td>
<td>46</td>
<td>3,658</td>
</tr>
<tr>
<td>make-3.76.1</td>
<td>27K</td>
<td>190</td>
<td>14,010</td>
<td>9,094</td>
<td>57</td>
<td>4,527</td>
</tr>
<tr>
<td>wget-1.9</td>
<td>35K</td>
<td>433</td>
<td>28,958</td>
<td>14,537</td>
<td>13</td>
<td>6,675</td>
</tr>
<tr>
<td>screen-4.0.2</td>
<td>45K</td>
<td>588</td>
<td>39,693</td>
<td>29,498</td>
<td>65</td>
<td>12,566</td>
</tr>
<tr>
<td>a2ps-4.14</td>
<td>64K</td>
<td>980</td>
<td>86,867</td>
<td>27,565</td>
<td>6</td>
<td>17,684</td>
</tr>
<tr>
<td>sendmail-8.13.6</td>
<td>130K</td>
<td>756</td>
<td>76,630</td>
<td>52,505</td>
<td>60</td>
<td>19,135</td>
</tr>
<tr>
<td>nethack-3.3.0</td>
<td>211K</td>
<td>2,207</td>
<td>237,427</td>
<td>157,645</td>
<td>997</td>
<td>54,989</td>
</tr>
<tr>
<td>vim60</td>
<td>227K</td>
<td>2,770</td>
<td>150,950</td>
<td>107,629</td>
<td>1,668</td>
<td>40,979</td>
</tr>
<tr>
<td>emacs-22.1</td>
<td>399K</td>
<td>3,388</td>
<td>204,865</td>
<td>161,118</td>
<td>1,554</td>
<td>66,413</td>
</tr>
<tr>
<td>python-2.5.1</td>
<td>435K</td>
<td>2,996</td>
<td>241,511</td>
<td>99,014</td>
<td>723</td>
<td>51,859</td>
</tr>
<tr>
<td>linux-3.0</td>
<td>710K</td>
<td>13,856</td>
<td>345,407</td>
<td>300,203</td>
<td>493</td>
<td>139,667</td>
</tr>
<tr>
<td>gimp-2.6</td>
<td>959K</td>
<td>11,728</td>
<td>1,482,230</td>
<td>286,588</td>
<td>2</td>
<td>190,806</td>
</tr>
<tr>
<td>ghostscript-9.00</td>
<td>1,363K</td>
<td>12,993</td>
<td>2,891,500</td>
<td>342,293</td>
<td>39</td>
<td>201,161</td>
</tr>
</tbody>
</table>

maxSCC: size of the largest recursive call cycle
Experiment Results

Buffer Overrun Detection with Intervals

<table>
<thead>
<tr>
<th>Programs</th>
<th>Interval\textsubscript{vanilla}</th>
<th>Interval\textsubscript{base}</th>
<th>Spd\textsuperscript{↑1}</th>
<th>Interval\textsubscript{sparse}</th>
<th>Spd\textsuperscript{↑2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip-1.2.4a</td>
<td>772</td>
<td>14</td>
<td>55 x</td>
<td>3</td>
<td>5 x</td>
</tr>
<tr>
<td>bc-1.06</td>
<td>1,270</td>
<td>96</td>
<td>13 x</td>
<td>7</td>
<td>14 x</td>
</tr>
<tr>
<td>tar-1.13</td>
<td>12,947</td>
<td>338</td>
<td>38 x</td>
<td>8</td>
<td>42 x</td>
</tr>
<tr>
<td>less-382</td>
<td>9,561</td>
<td>1,211</td>
<td>8 x</td>
<td>33</td>
<td>37 x</td>
</tr>
<tr>
<td>make-3.76.1</td>
<td>24,240</td>
<td>1,893</td>
<td>13 x</td>
<td>21</td>
<td>90 x</td>
</tr>
<tr>
<td>wget-1.9</td>
<td>44,092</td>
<td>1,214</td>
<td>36 x</td>
<td>11</td>
<td>110 x</td>
</tr>
<tr>
<td>screen-4.0.2</td>
<td>∞</td>
<td>31,324</td>
<td>N/A</td>
<td>767</td>
<td>41 x</td>
</tr>
<tr>
<td>a2ps-4.14</td>
<td>∞</td>
<td>3,200</td>
<td>N/A</td>
<td>40</td>
<td>80 x</td>
</tr>
<tr>
<td>sendmail-8.13.6</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>744</td>
<td>N/A</td>
</tr>
<tr>
<td>nethack-3.3.0</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>16,373</td>
<td>N/A</td>
</tr>
<tr>
<td>vim60</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>23,798</td>
<td>N/A</td>
</tr>
<tr>
<td>emacs-22.1</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>37,830</td>
<td>N/A</td>
</tr>
<tr>
<td>python-2.5.1</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>11,039</td>
<td>N/A</td>
</tr>
<tr>
<td>linux-3.0</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>33,618</td>
<td>N/A</td>
</tr>
<tr>
<td>gimp-2.6</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>3,874</td>
<td>N/A</td>
</tr>
<tr>
<td>ghostscript-9.00</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>14,814</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Interval\textsubscript{vanilla}: worklist-based analyzer with no optimization  
Interval\textsubscript{base}: Interval\textsubscript{vanilla} + localization optimization (VMCAI’11)  
Interval\textsubscript{sparse}: sparse version of Interval\textsubscript{vanilla}
Experiment Results

Buffer Overrun Detection with Intervals

<table>
<thead>
<tr>
<th>Programs</th>
<th>Interval_{vanilla}</th>
<th>Interval_{base}</th>
<th>Spd↑₁</th>
<th>Interval_{sparse}</th>
<th>Spd↑₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip-1.2.4a</td>
<td>772</td>
<td>14</td>
<td>55 x</td>
<td>3</td>
<td>5 x</td>
</tr>
<tr>
<td>bc-1.06</td>
<td>1,270</td>
<td>96</td>
<td>13 x</td>
<td>7</td>
<td>14 x</td>
</tr>
<tr>
<td>tar-1.13</td>
<td>12,947</td>
<td>338</td>
<td>38 x</td>
<td>8</td>
<td>42 x</td>
</tr>
<tr>
<td>less-382</td>
<td>9,561</td>
<td>1,211</td>
<td>8 x</td>
<td>33</td>
<td>37 x</td>
</tr>
<tr>
<td>make-3.76.1</td>
<td>24,240</td>
<td>1,893</td>
<td>13 x</td>
<td>21</td>
<td>90 x</td>
</tr>
<tr>
<td>wget-1.9</td>
<td>44,092</td>
<td>1,214</td>
<td>36 x</td>
<td>11</td>
<td>110 x</td>
</tr>
<tr>
<td>screen-4.0.2</td>
<td>∞</td>
<td>31,324</td>
<td>N/A</td>
<td>767</td>
<td>41 x</td>
</tr>
<tr>
<td>a2ps-4.14</td>
<td>∞</td>
<td>3,200</td>
<td>N/A</td>
<td>40</td>
<td>80 x</td>
</tr>
<tr>
<td>sendmail-8.13.6</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>744</td>
<td>N/A</td>
</tr>
<tr>
<td>nethack-3.3.0</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>16,373</td>
<td>N/A</td>
</tr>
<tr>
<td>vim60</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>23,798</td>
<td>N/A</td>
</tr>
<tr>
<td>emacs-22.1</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>37,830</td>
<td>N/A</td>
</tr>
<tr>
<td>python-2.5.1</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>11,039</td>
<td>N/A</td>
</tr>
<tr>
<td>linux-3.0</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>33,618</td>
<td>N/A</td>
</tr>
<tr>
<td>gimp-2.6</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>3,874</td>
<td>N/A</td>
</tr>
<tr>
<td>ghostscript-9.00</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>14,814</td>
<td>N/A</td>
</tr>
</tbody>
</table>

code size is not the only factor for the running time
Buffer Overrun Detection with Octagons

<table>
<thead>
<tr>
<th>Programs</th>
<th>Octagon_{vanilla}</th>
<th>Octagon_{base}</th>
<th>Spd↑₁</th>
<th>Octagon_{sparse}</th>
<th>Spd↑₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip-1.2.4a</td>
<td>2,078</td>
<td>273</td>
<td>8 x</td>
<td>21</td>
<td>13 x</td>
</tr>
<tr>
<td>bc-1.06</td>
<td>9,536</td>
<td>1,065</td>
<td>9 x</td>
<td>55</td>
<td>19 x</td>
</tr>
<tr>
<td>tar-1.13</td>
<td>∞</td>
<td>9,566</td>
<td>N/A</td>
<td>188</td>
<td>51 x</td>
</tr>
<tr>
<td>less-382</td>
<td>∞</td>
<td>16,121</td>
<td>N/A</td>
<td>432</td>
<td>37 x</td>
</tr>
<tr>
<td>make-3.76.1</td>
<td>∞</td>
<td>17,724</td>
<td>N/A</td>
<td>331</td>
<td>53 x</td>
</tr>
<tr>
<td>wget-1.9</td>
<td>∞</td>
<td>15,998</td>
<td>N/A</td>
<td>288</td>
<td>56 x</td>
</tr>
<tr>
<td>screen-4.0.2</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>16,433</td>
<td>N/A</td>
</tr>
<tr>
<td>a2ps-4.14</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>8,566</td>
<td>N/A</td>
</tr>
<tr>
<td>sendmail-8.13.6</td>
<td>∞</td>
<td>∞</td>
<td>N/A</td>
<td>64,808</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Octagon_{vanilla}: worklist-based analyzer with no optimization
Octagon_{base}: Octagon_{vanilla} + localization optimization
Octagon_{sparse}: sparse version of Octagon_{vanilla}
Sparse analysis makes sound static analysis scalable
Questions?

for technical details: