Static Analysis Research @KU

Hakjoo Oh
Programmin Research Laboratory
Korea Univeristy

2016.06
Motivation: Unsafe Software
Motivation: Unsafe Software
Motivation: Unsafe Software
Motivation: Unsafe Software
Motivation: Unsafe Software
Current Technology for Safe SW

Manual, ad-hoc, postmortem:

code review, testing, simulation, debugging, etc
Our Mission

Technology for "Software MRI"
Aims to detect memory errors in C programs
- e.g., buffer-overrun, memory leak, null-dereference, etc

Features (vs. testing)
- Full automation
- Find bugs early
- All bugs found
static char *curfinal = "HDACB FE";

keysym = read_from_input();

if (((((KeySym)(keysym) >= 0xFF91) && ((KeySym)(keysym) <= 0xFF94)))
    {
        unparseputc((char)(keysym-0xFF91 + 'P'), pty);
        key = 1;
    }
else if (keysym >= 0)
    {
        if (keysym < 16)
        {
            if (read_from_input())
            {
                if (keysym >= 10) return;
                curfinal[keysym] = 1;
            }
            else
            {
                curfinal[keysym] = 2;
            }
        }
        if (keysym < 10)
        {
            unparseputc(curfinal[keysym], pty);
        }
    }
static char *curfinal = "HDACB FE";

keysym = read_from_input ();

if (((((KeySym)(keysym) >= 0xFF91) && ((KeySym)(keysym) <= 0xFF94)))
{
    unparseputc((char)(keysym-0xFF91 + 'P'), pty);
    key = 1;
}
else if (keysym >= 0)
{
    if (keysym < 16)
    {
        if (read_from_input())
        {
            if (keysym >= 10) return;
            curfinal[keysym] = 1;
        }
        else
        {
            curfinal[keysym] = 2;
        }
    }
    if (keysym < 10)
    {
        unparseputc(curfinal[keysym], pty);
    }
}
static char *curfinal = "HDACB FE";

keysym = read_from_input ();

if (((((KeySym)(keysym) >= 0xFF91) && ((KeySym)(keysym) <= 0xFF94))
{
    unparseputc((char)((keysym-0xFF91) + 'P'), pty);
    key = 1;
}
else if (keysym >= 0)
{
    if (keysym < 16)
    {
        if (read_from_input())
        {
            if (keysym >= 10) return;
            curfinal[keysym] = 1;
        }
        else
        {
            curfinal[keysym] = 2;
        }
    }
    if (keysym < 10)
    {
        unparseputc(curfinal[keysym], pty);
    }
}
static char *curfinal = "HDACB FE";

keysym = read_from_input();

if (((((KeySym)(keysym)) >= 0xFF91) && ((KeySym)(keysym)) <= 0xFF94))
{
    unparseputc((char)(keysym-0xFF91 + 'P'), pty);
    key = 1;
}
else if (keysym >= 0)
{
    if (keysym < 16)
    {
        if (read_from_input())
        {
            if (keysym >= 10) return;
            curfinal[keysym] = 1;
        }
        else
        {
            curfinal[keysym] = 2;
        }
    }
    else
    {
        curfinal[keysym] = 1;
    }
}
else if (keysym < 10)
{
    unparseputc(curfinal[keysym], pty);
}

Sparrow automatically pinpoints the buffer-overflow bug
static char *curfinal = "HDACB  FE";

keysym = read_from_input();

if (((((KeySym)(keysym) >= 0xFF91) && ((KeySym)(keysym) <= 0xFF94)))
{
    unparseputc((char)(keysym-0xFF91 + 'P'), pty);
    key = 1;
}
else if (keysym >= 0)
{
    if (keysym < 16)
    {
        if (read_from_input())
        {
            if (keysym >= 10) return;
            curfinal[keysym] = 1;
        }
        else
        {
            curfinal[keysym] = 2;
        }
    }
    if (keysym < 10)
    {
        unparseputc(curfinal[keysym], pty);
    }
}
static char *curfinal = "HDACB FE";

keysym = read_from_input();

if (((((KeySym)(keysym) >= 0xFF91) && (keysym) <= 0xFF94)))
{
    unparseputc((char)(keysym-0xFF91 + 'P'), pty);
    key = 1;
}
else if (keysym >= 0)
{
    if (keysym < 16)
    {
        if (read_from_input())
        {
            if (keysym >= 10) return;
            curfinal[keysym] = 1;
        }
        else
        {
            curfinal[keysym] = 2;
        }
    }
    if (keysym < 10)
    {
        unparseputc(curfinal[keysym], pty);
    }
}

Sparrow automatically pinpoints the buffer-overrun bug
Static Program Analysis

• Predict SW behavior statically and automatically

  • static: before execution, before sell / embed
  • automatic: sw is analyzed by sw (“static analyzers”)

• Applications
Static Program Analysis

• Predict SW behavior statically and automatically
  • **static**: before execution, before sell / embed
  • **automatic**: sw is analyzed by sw (“static analyzers”)

• Applications
  • **bug-finding**: e.g., find runtime failures of programs
Static Program Analysis

• Predict SW behavior statically and automatically
  • **static**: before execution, before sell / embed
  • **automatic**: sw is analyzed by sw (“static analyzers”)

• Applications
  • **bug-finding**: e.g., find runtime failures of programs
  • **security**: e.g., is this app malicious or benign?
Static Program Analysis

• Predict SW behavior statically and automatically
  • static: before execution, before sell / embed
  • automatic: sw is analyzed by sw (“static analyzers”)
• Applications
  • bug-finding. e.g., find runtime failures of programs
  • security. e.g., is this app malicious or benign?
  • verification. e.g., does the program meet its specification?
Static Program Analysis

• Predict SW behavior statically and automatically
  • static: before execution, before sell / embed
  • automatic: sw is analyzed by sw (“static analyzers”)
• Applications
  • bug-finding. e.g., find runtime failures of programs
  • security. e.g., is this app malicious or benign?
  • verification. e.g., does the program meet its specification?
  • compiler optimization, e.g., automatic parallelization
We have a number of research internship openings in 2015. Internships could be part-time or full-time and offer great opportunity for any students thinking about doing research in program analysis. Possible topics include:

- Dynamic symbolic execution
- Refinement-based alias analysis
- Distributed static analysis of large applications
- Identification of vulnerabilities in Java through static analysis
- Concurrent data-flow analysis
- Refining flow-insensitive analyses

Ideal candidates would have strong research background and solid (C++) programming skills.
A tool to detect bugs in Android and iOS apps before they ship
Static Code Analysis

Potential bugs—in the source code of a project with the static analyzer built into Xcode. Source code may have subtle errors that the compiler and manifest themselves only at runtime, when they could be difficult to identify and fix.

Steps
1. Choose Product > Analyze.
2. In the issue navigator, select an analyzer message.
3. In the source editor, click the corresponding message.
4. Use the pop-up menu in the analysis results bar above the edit area to study the flow path of the flaw.
5. Edit the code to fix the flaw.

The video shows the process of looking at a flaw in the source file SKTText.m.
Mobile Security

Detect and prevent cyber attacks that spy on, profile, or use mobile devices

Malicious apps compromise mobile security to access private information, such as contact lists and calendar details. They also use mobile device features, such as cameras and microphones, to spy, profile users, or conduct cyber attacks.

FireEye Mobile Security (Mobile Threat Prevention) detects and prevents these mobile threats and provides visibility into mobile security trends across the enterprise. FireEye Mobile Threat Prevention also integrates with industry leading mobile device management (MDM) providers.
Our Research

Towards **sound**, **precise**, and **scalable** static analysis

- Sparse analysis framework
- Selective X-Sensitivity
- Machine learning techniques
(I) Soundness

Find all bugs / verify absence
(1) Soundness

Find all bugs / verify absence
(1) Soundness

Find all bugs / verify absence
(2) Precision

Few false alarms
(3) Scalability

Large programs

nethack-3.3.0 (211KLoC)
Challenge in Static Analysis

Soundness

Scalability

Precision
Challenge in Static Analysis

Soundness

Scalability

Precision

“bug-finders”
Challenge in Static Analysis

Soundness

“verifiers”

Scalability

Precision

“bug-finders”
Challenge in Static Analysis

Soundness

“optimizers”

“verifiers”

Scalability

Precision

“bug-finders”
Our Research Goal

Soundness

Scalability        !        Precision
Our Research Goal

General Sparse Analysis Framework
Our Research Goal

- Soundness
- Scalability
- Precision

General Sparse Analysis Framework

Selective X-Sensitivity, Adaptation via ML
Significance

- Cracked down the common sense that sound, precise, and scalable static analysis is infeasible

- Publication:
  - General Sparse Analysis Framework
    - ACM PLDI 2012 (top conference in programming languages)
    - ACM TOPLAS 2014 (top journal in programming languages)
  - Selective X-Sensitivity Approach
    - ACM PLDI 2014 (top conference in programming languages)
    - ACM OOPSLA 2015 (top conference in programming languages)
    - ACM TOPLAS 2016 (top journal in programming languages)
Motivation

- Commercialized version of Sparrow
  - memory-bug-finding tool for full C
  - sound in design, unsound yet scalable in reality
- Realistic workbench available
  - “let’s try to achieve sound, precise, yet scalable version”
The Challenge in Reality

Soundness

Scalability

Precision

(sound- &- global version)

(sound- &- global version)
The First Goal: Scalability

Soundness

Scalability

Precision

(2012, sound-&-global version)
Scalability Improvement

(2012, sound-&-global version)

1 Million LoC in 10 hrs

1.3 M Scalability (LOC)
600x Scalability (speed)
350x
460x
300K
900K
1 Million LoC in 10 hrs
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis

“Right Part at Right Moment”

```plaintext
\[
\begin{align*}
x & = x + 1 \\
y & = y - 1 \\
z & = x \\
v & = y \\
\text{ret} & = *a + *b
\end{align*}
\]
### Key: General Sparse Analysis

"Right Part at Right Moment"

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>v</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \(x = x + 1\)
- \(y = y - 1\)
- \(z = x\)
- \(v = y\)
- \(\text{ret} \times a + \times b\)
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis
“Right Part at Right Moment”
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis
“Right Part at Right Moment”

x = x + 1
y = y - 1
z = x
v = y
ret *a + *b
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis

“Right Part at Right Moment”

\begin{align*}
x &= x + 1 \\
y &= y - 1 \\
z &= x \\
v &= y \\
\text{ret } *a + *b
\end{align*}
Key: General Sparse Analysis

“Right Part at Right Moment”

\[
x = x + 1
\]

\[
y = y - 1
\]

\[
z = x
\]

\[
v = y
\]

\[
\text{ret} \ast a + \ast b
\]
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis

“Right Part at Right Moment”
Key: General Sparse Analysis

“Right Part at Right Moment”
General Sparse Analysis Framework

Theorem. (preservation of soundness and precision)

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

\[ \text{fix } \hat{F} = \text{fix } \hat{F}_s \]

“An important strength is that the theoretical result is very general ... The result should be highly influential on future work in sparse analysis.” (from PLDI reviews)
The Second Goal: Precision

Soundness

Scalability  Precision
The Second Goal: Precision

Soundness

Scalability

Precision

**Challenge**: Can we achieve it without scalability loss?
cf) Existing Techniques

- Soundness
- Scalability
- Precision

context-sensitivity, relational analysis, etc
Selective X-Sensitivity Approach

• **Key Idea**: Improve precision only when it matters
Effectiveness

+25% / -25%

vs.

+25% / -1300%
Flow-Sensitivity

\[
\begin{align*}
x &= y = 0; z = 1 \\
x &= z \\
z &= z + 1 \\
y &= x \\
\text{assert}(y > 0)
\end{align*}
\]
Flow-Sensitivity

\[ x=y=0; z=1 \]

\[ x=z \]

\[ z=z+1 \]

\[ y=x \]

\[ \text{assert}(y>0) \]

\[
\begin{array}{|c|c|}
\hline
x & [0,0] \\
\hline
y & [0,0] \\
\hline
z & [1,1] \\
\hline
\end{array}
\]
Flow-Sensitivity

x=y=0; z=1

x=z

z=z+1

y=x

assert(y>0)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[1,1]</td>
</tr>
<tr>
<td></td>
<td>[1,1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1,1]</td>
<td>[0,0]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1,1]</td>
<td></td>
<td>[1,1]</td>
</tr>
</tbody>
</table>
Flow-Sensitivity

\[ x = y = 0; z = 1 \]

\[ z = z + 1 \]

\[ y = x \]

\[ \text{assert}(y > 0) \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>( y )</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>( z )</td>
<td>[1, 1]</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>( y )</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>( z )</td>
<td>[1, 1]</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>( y )</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>( z )</td>
<td>[2, 2]</td>
</tr>
</tbody>
</table>
Flow-Sensitivity

x = y = 0; z = 1

x = z

z = z + 1

y = x

assert(y > 0)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[1,1]</td>
</tr>
<tr>
<td>2</td>
<td>[1,1]</td>
<td>[0,0]</td>
<td>[1,1]</td>
</tr>
<tr>
<td>3</td>
<td>[1,1]</td>
<td>[0,0]</td>
<td>[2,2]</td>
</tr>
<tr>
<td>4</td>
<td>[1,1]</td>
<td>[1,1]</td>
<td>[2,2]</td>
</tr>
</tbody>
</table>
Flow-Sensitivity

\[ x = y = 0; z = 1 \]
\[ x = z \]
\[ z = z + 1 \]
\[ y = x \]
\[ \text{assert}(y > 0) \]

<table>
<thead>
<tr>
<th></th>
<th>( x )</th>
<th>( y )</th>
<th>( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[1,1]</td>
</tr>
<tr>
<td>2</td>
<td>[1,1]</td>
<td>[0,0]</td>
<td>[1,1]</td>
</tr>
<tr>
<td>3</td>
<td>[1,1]</td>
<td>[0,0]</td>
<td>[2,2]</td>
</tr>
<tr>
<td>4</td>
<td>[1,1]</td>
<td>[1,1]</td>
<td>[2,2]</td>
</tr>
</tbody>
</table>

precise but costly
Flow-Insensitivity

\[ x = y = 0; z = 1 \]

\[ x = z \]

\[ z = z + 1 \]

\[ y = x \]

\[ \text{assert}(y > 0) \]

<table>
<thead>
<tr>
<th></th>
<th>([0, + \infty])</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td></td>
</tr>
<tr>
<td>(y)</td>
<td>([0, + \infty])</td>
</tr>
<tr>
<td>(z)</td>
<td>([1, + \infty])</td>
</tr>
</tbody>
</table>
Flow-Insensitivity

\[ x = y = 0; z = 1 \]

\[ x = z \]

\[ z = z + 1 \]

\[ y = x \]

assert(y > 0)

cheap but imprecise

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>([0, +\infty])</td>
</tr>
<tr>
<td>(y)</td>
<td>([0, +\infty])</td>
</tr>
<tr>
<td>(z)</td>
<td>([1, +\infty])</td>
</tr>
</tbody>
</table>
Selective Flow-Sensitivity

\[ x=y=0; z=1 \]
\[ x=z \]
\[ z=z+1 \]
\[ y=x \]
\[ \text{assert}(y>0) \]

FS : \{x\}  
FI : \{y,z\}
Selective Flow-Sensitivity

\[ x = y = 0; z = 1 \]
\[ x = z \]
\[ z = z + 1 \]
\[ y = x \]
\[ \text{assert}(y > 0) \]

**FS**: \{\(x\)\}

<table>
<thead>
<tr>
<th>(x)</th>
<th>[0,0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>[1,(+\infty)]</td>
</tr>
<tr>
<td>(x)</td>
<td>[1,(+\infty)]</td>
</tr>
<tr>
<td>(x)</td>
<td>[1,(+\infty)]</td>
</tr>
</tbody>
</table>

**FI**: \{\(y, z\)\}

| \(y\) | [0,\(+\infty\)] |
| \(z\) | [1,\(+\infty\)] |
Selective Flow-Sensitivity

\[
x = y = 0; z = 1
\]
\[
x = z
\]
\[
z = z + 1
\]
\[
y = x
\]
\[
\text{assert}(y > 0)
\]

FS : \{x\}

<table>
<thead>
<tr>
<th></th>
<th>[0,0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>[1,(\infty)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>[1,(\infty)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>[1,(\infty)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

FI : \{y,z\}

<table>
<thead>
<tr>
<th></th>
<th>[0,(\infty)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>[1,(\infty)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

fail to prove
Selective Flow-Sensitivity

\[
\begin{align*}
x &= y = 0; z = 1 \\
x &= z \\
z &= z + 1 \\
y &= x \\
\text{assert}(y > 0)
\end{align*}
\]

FS : \{y\} \quad \text{FI : \{x,z\}}
Selective Flow-Sensitivity

$$x = y = 0; z = 1$$

$$x = z$$

$$z = z + 1$$

$$y = x$$

assert($$y > 0$$)

FS: \{y\}

<table>
<thead>
<tr>
<th>y</th>
<th>[0,0]</th>
</tr>
</thead>
</table>

FI: \{x, z\}

<table>
<thead>
<tr>
<th>x</th>
<th>[0, +\infty]</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>[1, +\infty]</td>
</tr>
</tbody>
</table>

fail to prove
Selective Flow-Sensitivity

\[ x = y = 0 ; z = 1 \]
\[ x = z \]
\[ z = z + 1 \]
\[ y = x \]
\[ \text{assert}(y > 0) \]

FS : \{z\}

<table>
<thead>
<tr>
<th>z</th>
<th>[1,1]</th>
</tr>
</thead>
</table>

FI : \{x,y\}

<table>
<thead>
<tr>
<th>x</th>
<th>[0, +\infty]</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>[0, +\infty]</td>
</tr>
</tbody>
</table>

fail to prove
Selective Flow-Sensitivity

FS : \{y, z\}  \quad  \text{FI} : \{x\}

x = y = 0; z = 1

x = z

z = z + 1

y = x

assert (y > 0)
Selective Flow-Sensitivity

```
x=y=0; z=1
```

```
x=z
```

```
z=z+1
```

```
y=x
```

```
assert(y>0)
```

**FS : \{y,z\}**

<table>
<thead>
<tr>
<th>y</th>
<th>[0,0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>[1,1]</td>
</tr>
</tbody>
</table>

**FI : \{x\}**

| x   | [0, +\infty] |

```
y   [0,0]   
z   [2,2]   
```

```
y   [0, +\infty]   
z   [2,2]   
```
Selective Flow-Sensitivity

\[ x = y = 0 ; z = 1 \]

\[ x = z \]

\[ z = z + 1 \]

\[ y = x \]

assert (y > 0)

FS : \{y, z\}

<table>
<thead>
<tr>
<th>y</th>
<th>[0, 0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>[1, 1]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y</th>
<th>[0, 0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>[1, 1]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y</th>
<th>[0, 0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>[2, 2]</td>
</tr>
</tbody>
</table>

Fl : \{x\}

| x   | [0, +\infty] |

fail to prove
Selective Flow-Sensitivity

\(x = y = 0; z = 1\)

\(x = z\)

\(z = z + 1\)

\(y = x\)

assert\((y > 0)\)

FS : \{x, y\}

<table>
<thead>
<tr>
<th></th>
<th>([0,0])</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

FI : \{z\}

<table>
<thead>
<tr>
<th></th>
<th>([1,\infty])</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

Succeed
Hard Search Problem

• Intractably large space, if not infinite
  • $2^{\text{Var}}$ different abstractions for FS
• Most of them are too imprecise or costly
  • $P(\{x,y,z\}) = \{\emptyset, \{x\}, \{y\}, \{z\}, \{x,y\}, \{y,z\}, \{x,z\}, \{x,y,z\}\}$
Our Solutions

• Two approaches:
  • PL approaches [PLDI’14, TOPLAS’16]
  • ML approaches [OOPSLA’15, on-going work]
Learning-based Approach
Learning-based Approach

• Parameterized adaptation strategy

\[ S_w : \text{pgm} \rightarrow 2^{\text{Var}} \]
Learning-based Approach

• Parameterized adaptation strategy

\[ S_w : \text{pgm} \to 2^{\text{Var}} \]

• Learn a good parameter \( W \) from existing codebase

\[
\begin{array}{c}
P_1, P_2, \ldots, P_m \\
\Rightarrow \\
W
\end{array}
\]

Codebase
Learning-based Approach

• Parameterized adaptation strategy

\[ S_w : \text{pgm} \rightarrow 2^{\text{Var}} \]

• Learn a good parameter \( W \) from existing codebase

\[
\begin{array}{c}
\text{P}_1, \text{P}_2, \ldots, \text{P}_m \\
\text{Codebase}
\end{array} \Rightarrow W
\]

• For new program \( P \), run static analysis with \( S_w(P) \)
1. Parameterized Strategy

\[ S_w : \text{pgm} \rightarrow 2^{\text{Var}} \]

(1) Represent program variables as feature vectors.

(2) Compute the score of each variable.

(3) Choose the top-k variables based on the score.
(1) Features

- Predicates over variables:
  \[ f = \{f_1, f_2, \ldots, f_5\} \quad (f_i: \text{Var} \rightarrow \{0,1\}) \]

- 45 simple syntactic features for variables: e.g,
  - local / global variable, passed to / returned from malloc, incremented by constants, etc

- Represent each variable as a feature vector:
  \[ f(x) = \langle f_1(x), f_2(x), f_3(x), f_4(x), f_5(x) \rangle \]
(2) Scoring

• The parameter $\mathbf{w}$ is a real-valued vector: e.g.,

$$\mathbf{w} = \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle$$

• Compute scores of variables:

$$\text{score}(x) = \langle 1, 0, 1, 0, 0 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.3$$
$$\text{score}(y) = \langle 1, 0, 1, 0, 1 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.6$$
$$\text{score}(z) = \langle 0, 0, 1, 1, 0 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.1$$
(3) Choose Top-k Variables

• Choose the top-k variables based on their scores: e.g., when k=2,

\[
\begin{align*}
\text{score}(x) &= 0.3 \\
\text{score}(y) &= 0.6 \\
\text{score}(z) &= 0.1
\end{align*}
\]

\(\{x, y\}\)

• In experiments, we chosen 10% of variables with highest scores.
2. Learn a Good Parameter

\[
\begin{align*}
P_1, P_2, \ldots, P_m & \quad \Rightarrow \quad W \\
\text{Codebase} & \\
\end{align*}
\]

- Solve the optimization problem:

Find \( w \) that maximizes

\[
\sum_{P_i} F(P_i, S_w(P_i))
\]
Solving the Opt. Problem

• How to solve the optimization problem efficiently?

Find \( w \) that maximizes \( \sum_{P_i} F(P_i, S_w(P_i)) \)

• Using ideas of Bayesian optimization

![Graph showing comparison between random sampling and Bayesian optimization](image)
Effectiveness

• Implemented in Sparrow, an interval analyzer for C
• Evaluated on 30 open-source benchmarks
Effectiveness

- Implemented in Sparrow, an interval analyzer for C
- Evaluated on 30 open-source benchmarks

Precision

<table>
<thead>
<tr>
<th>FL</th>
<th>SFS</th>
<th>FS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>70</td>
<td>100</td>
</tr>
</tbody>
</table>
Effectiveness

- Implemented in Sparrow, an interval analyzer for C
- Evaluated on 30 open-source benchmarks

### Precision

<table>
<thead>
<tr>
<th>FI</th>
<th>SFS</th>
<th>FS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>70</td>
<td>100</td>
</tr>
</tbody>
</table>

### Cost

<table>
<thead>
<tr>
<th>FI</th>
<th>SFS</th>
<th>FS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x</td>
<td>2x</td>
<td>18x</td>
</tr>
</tbody>
</table>
• The success crucially depends on the choice of features
• Feature construction is nontrivial and tedious
• \(|\text{analyzers}| \times |\text{parameter types}| \times |\text{query types}|\)

<table>
<thead>
<tr>
<th>Type</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>leaf function</td>
</tr>
<tr>
<td></td>
<td>function containing malloc</td>
</tr>
<tr>
<td></td>
<td>function containing realloc</td>
</tr>
<tr>
<td></td>
<td>function containing a loop</td>
</tr>
<tr>
<td></td>
<td>function containing an if statement</td>
</tr>
<tr>
<td></td>
<td>function containing a switch statement</td>
</tr>
<tr>
<td></td>
<td>function using a string-related library function</td>
</tr>
<tr>
<td></td>
<td>write to a global variable</td>
</tr>
<tr>
<td></td>
<td>read a global variable</td>
</tr>
<tr>
<td></td>
<td>write to a structure field</td>
</tr>
<tr>
<td></td>
<td>read from a structure field</td>
</tr>
<tr>
<td></td>
<td>directly return a constant expression</td>
</tr>
<tr>
<td></td>
<td>indirectly return a constant expression</td>
</tr>
<tr>
<td></td>
<td>directly return an allocated memory</td>
</tr>
<tr>
<td></td>
<td>indirectly return an allocated memory</td>
</tr>
<tr>
<td></td>
<td>directly return a reallocated memory</td>
</tr>
<tr>
<td></td>
<td>indirectly return a reallocated memory</td>
</tr>
<tr>
<td></td>
<td>return expression involves field access</td>
</tr>
<tr>
<td></td>
<td>return value depends on a structure field</td>
</tr>
<tr>
<td></td>
<td>return void</td>
</tr>
<tr>
<td></td>
<td>directly invoked with a constant</td>
</tr>
<tr>
<td></td>
<td>constant is passed to an argument</td>
</tr>
<tr>
<td></td>
<td>invoked with an unknown value</td>
</tr>
<tr>
<td></td>
<td>functions having no arguments</td>
</tr>
<tr>
<td></td>
<td>functions having one argument</td>
</tr>
<tr>
<td></td>
<td>functions having more than one argument</td>
</tr>
<tr>
<td></td>
<td>functions having an integer argument</td>
</tr>
<tr>
<td></td>
<td>functions having a pointer argument</td>
</tr>
<tr>
<td></td>
<td>functions having a structure as an argument</td>
</tr>
</tbody>
</table>

Hurdle
Learning with Automatic Feature Construction

• Fully automatic learning approach

\[ P_1, P_2, \ldots, P_m \] → Features → Training Examples → \[ C : P \rightarrow B \]
Powerful Static Analysis Enabled

General Sparse Analysis Framework

Selective X-Sensitivity, Adaptation via ML

Soundness

Scalability

Precision

Sparrow
The Early Bird

Copyright © 2007 Faso.com, Inc. All rights reserved.
Applications (1)

Security/Safety-critical softwares

Massive Security Bug In OpenSSL Could Affect A Huge Chunk Of The Internet

I saw a t-shirt one time, “I’m a bomb disposal technician,” it read. “If you see me running, try to keep up.”

The same sort of idea can be applied to network security: when all the net security people you know are freaking out, it’s probably an okay time to worry.

This afternoon, many of the net security people I know are freaking out. A very serious bug in OpenSSL — a cryptographic library that

Sendmail disasters

These are the most serious sendmail security and reliability problems through sendmail 8.8.7 in 1997. Unattributed quotes here are from Ali
Applications (2)

Efficient uses of modern computing platforms

- Mobile
- Cloud
- Parallel
- Wearable
- …

• New Software challenges:
  e.g., reliability, energy-efficiency, security, …
Static Analysis for Mobile Computing
Static Analysis for Mobile Computing

B: compute-intensive part
Static Analysis for Mobile Computing

**Plan:** Static analysis to estimate power consumption

B: compute-intensive part
Applications (3)

Program synthesis (automatic programming)

```java
public int enqueue(Queue q, int i)
{  
  Node n = new Node();
  n.val = i;
  n.next = null;
  if(q.head != null){
    q.tail.next = n;
  }
  if(q.head == null){
    q.head = n;
  }
  q.tail = n;
  return 1;
}
```
Regular Expressions

- A formal language for expressing string patterns

\[ e \rightarrow a \in \Sigma \mid \epsilon \mid \emptyset \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e^* \]
Automatic Synthesis of Regular Expressions

 옳은 예 (Positive examples)
00,
1001,
0101001010
1111001111

틀린 예 (Negative examples)
01,
11,
000,
00100

자동 합성된 정규 표현식
(0?1)*00(10?)*
(in 0.5s)
Automatic Synthesis of Regular Expressions

- Search-based program synthesis

\[ s \to a \in \Sigma \mid \epsilon \mid \emptyset \mid s_1 + s_2 \mid s_1 \cdot s_2 \mid s^* \mid \square \]
Automatic Synthesis of Regular Expressions

- Pruning search space in sound ways

1. 간단한 정규식 우선탐색 (Best-first Enumerative Search)

\[
C(a) = C(\epsilon) = C(\emptyset) = c_1 \\
C(\square) = c_2 \ (c_2 > c_1) \\
C(e_1 + e_2) > C(e_1) + C(e_2) \\
C(e_1 \cdot e_2) > C(e_1) + C(e_2) \\
C(e^*) > C(e)
\]

2. 같은 의미 상태 (Semantically-Equivalent States) 가지치기

\[
[s^* s^*] = [s^*] \\
[(s + s)] = [s] \\
[((s \cdot s^*)^*)] = [s^*]
\]

3. 해를 가질 수 없는 상태 (Dead States) 가지치기

\[
(b \in \mathcal{P}) \\
(a \in \mathcal{N})
\]

4. 불필요한 상태 (Redundant States) 가지치기

\[
(aab \in \mathcal{P})
\]
Automatic Synthesis of Regular Expressions

고안한 가지치기 기법들은 프로그래밍 언어 이론에 기반하여 결과의 안전성(Soundness)을 보장.

**Definition 1 (Dead States).** Let \((\mathcal{P}, \mathcal{N})\) be a regular expression problem. We say a state \(s \in S\) is dead, denoted \(\text{dead}(s)\), iff every closed state \(s'\) reachable from \(s\) is not a solution:

\[
\text{dead}(s) \iff (s \rightarrow^* s') \land s' \nrightarrow \rightarrow \neg \text{solution}(s').
\]

**Lemma 4.** Let \(s\) be any state. Then,

\[
\text{pdead}(s) \iff \exists p \in \mathcal{P}. p \not\in [s].
\]

*Proof.* Consider each direction.

- \((\Rightarrow)\) Suppose \(\text{pdead}(s)\) holds:

\[
s \rightarrow^* s' \land s' \nrightarrow \rightarrow \exists p \in \mathcal{P}. p \not\in [s'].
\]  (5)

From (5) and Lemma 6, we obtain \(\exists p \in \mathcal{P}. p \not\in [s]\).

- \((\Leftarrow)\) Suppose \(p \not\in [s]\). By Lemma 2, we have

\[
p \not\in \bigcup_{s \rightarrow^* s'} [s'].
\]

which implies that \(p \not\in [s']\) for all closed \(s'\) reachable from \(s\).

**Lemma 5.** Let \(s\) be any state. Then,

\[
\neg \text{dead}(s) \iff \exists n \in \mathcal{N}. n \in [s].
\]

*Proof.* Consider each direction.

- \((\Rightarrow)\) Suppose \(\neg \text{dead}(s)\) holds:

\[
s \rightarrow^* s' \land s' \nrightarrow \rightarrow \exists n \in \mathcal{N}. n \in [s'].
\]  (6)

From (6) and Lemma 7, we obtain \(\exists n \in \mathcal{N}. n \in [s]\).

- \((\Leftarrow)\) Suppose \(n \in [s]\). By Lemma 3, we have

\[
n \in \bigcap_{s \rightarrow^* s' \land s' \nrightarrow \rightarrow} [s']
\]

which implies that \(n \in [s']\) for all closed \(s'\) reachable from \(s\).
## Automatic Synthesis of Regular Expressions

<table>
<thead>
<tr>
<th>문제</th>
<th>예제 개수</th>
<th>합성된 정규식</th>
<th>소요 시간 (초)</th>
<th>속도향상</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>N</td>
<td></td>
<td>기본 알고리즘</td>
</tr>
<tr>
<td>w는 오른쪽으로부터 5번째 글자가 1이다.</td>
<td>3</td>
<td>3</td>
<td>(0+1)*1(0+1)(0+1)(0+1)(0+1)</td>
<td>148.0</td>
</tr>
<tr>
<td>w는 최대 두 개의 0을 가진다.</td>
<td>8</td>
<td>7</td>
<td>1<em>0?1</em>0?1*</td>
<td>425.0</td>
</tr>
<tr>
<td>w는 0과 1이 번갈아가며 등장한다.</td>
<td>10</td>
<td>11</td>
<td>0?(10)*1?</td>
<td>4073.9</td>
</tr>
<tr>
<td>w에 있는 0의 개수는 3으로 나누어 떨어진다.</td>
<td>8</td>
<td>7</td>
<td>(1+01<em>01</em>0)*</td>
<td>&gt; 7200.0</td>
</tr>
<tr>
<td>w가 0으로 시작하면 홀수의 길이를 가지고, 1로 시작하면 짝수의 길이를 가진다.</td>
<td>5</td>
<td>3</td>
<td>(0+1(0+1)) ((0+1)(0+1))*</td>
<td>&gt; 7200.0</td>
</tr>
<tr>
<td>w는 최소 1개의 0과 최대 1개의 1을 가진다.</td>
<td>12</td>
<td>10</td>
<td>0*(01?+100*)</td>
<td>&gt; 7200.0</td>
</tr>
<tr>
<td>w는 최대 1쌍의 연속한 1을 가진다.</td>
<td>9</td>
<td>8</td>
<td>(1+(01?)<em>)(0+10</em>)</td>
<td>465.1</td>
</tr>
</tbody>
</table>
Summary

Program Synthesis

Powerful Static Analysis Technology

SW Security & Verification

Modern Computing

Programming Languages Theories