Using Likely Invariants for Automated Software Fault Localization

Research by Swarup Kumar Sahoo, John Criswell, Chase Geigle, and Vikram Adve
Department of Computer Science at UIUC

presenter name(s) removed for FERPA considerations
Introduction.
Contributions.

- A novel invariant based approach for fault localization.
- A novel heuristic for reducing false positives in the diagnosis results.
- Evaluate their approach against a set of applications much bigger and more realistic than most previous work.
- Results show that their approach is effective at reducing root causes in large programs. Also, each step of filtering is important in the reduction of the set.
Key Idea.

- Invariants that are “similar” to training inputs are more effective than other types of invariants generated.
- Sophisticated filtering techniques allow us to start with a large number of suspected bug locations and let us narrow it down to a much smaller number of locations.
Figure 1. Diagnosis Tool Architecture
Provided Example

Fails on inputs with (year=0, month<=2).

A buffer overflow occurs at Line 31 when `type_names` is indexed with `weekday`, a negative value on said failed input.

A brief explanation...
Provided Example

Fails on inputs with (year=0, month<=2).

year is of type uint, and when decremented from value ‘0’, becomes the maximum unsigned value ($2^b - 1$) due to modular wraparound...
Provided Example

Fails on inputs with \((\text{year}=0, \text{ month} \leq 2)\).

...so \text{temp} becomes huge...
Provided Example

Fails on inputs with \((\text{year}=0, \text{ month} \leq 2)\).

...and \text{calc\_daynr} returns a very negative value...
Provided Example

Fails on inputs with $(\text{year}=0, \text{month}\leq2)$. …and this negative value carries through to the index of type_names, causing the buffer overflow.
Provided Example

Fails on inputs with \((\text{year}=0, \text{month}\leq2)\).

The buffer overflow occurs at Line 31 but the root cause is at Line 10.
The system needs to be provided:

1. A program
2. A specification for valid inputs (tokenizer or lexical analyzer)
3. A set of failing “bad” inputs that expose the bug in the program
Phase 1 generates a large set of “candidate” root causes. The approach minimizes false-negatives but gets a lot of false-positives. Phase 2 uses sophisticated filtering to cut down on the many false-positives.

Why? It is important not to miss a root cause from the very beginning.
How does it work? Generating Inputs

Based on what the user provided, a set of passing “good” inputs that do not expose the bug in the program is generated.

The “good” inputs are generated to be lexicographically close to the failing inputs.

“Bad” (year=0, month=2) “Good” (year=1, month=3)
The provided program is executed with the “good” inputs that were just generated, and a set of “narrow” range invariants is derived.

- **year** is $\geq 0$
- **month** is in the range $[0, +12]$
- **calc_daynr** returns $\geq 0$
- **calc_weekday** returns $\geq 0$

The invariant sets are limited to load, store, and function return values.

*Figure 1. Diagnosis Tool Architecture*
“Bad” traces are evaluated for violated range invariants.

In the example earlier, the range invariant (year $\geq 0$) is violated for the provided bad inputs.

However, there are also 94 other invariants that were violated. These need to be filtered!
How does it work? Dynamic Backwards Slicing

DBS takes as input the buggy statement, i.e. Line 31 `type_names[weekday]`

DBS then uses data flow and control flow to find invariant statements in the execution that affected the buggy statement.

These form the Dynamic Backward Slice. All the rest are filtered out!
How does it work? Dependence Filtering

Evaluates the data flow and control dependence graphs to filter out any statements with violated invariants that depend on another statement with violated invariants.

The candidate `calc_weekday` is dependent on another candidate `calc_weekday`.

```plaintext
29    weekday = calc_weekday(calc_daynr(t->year, t->month, t->day), 0);
30    str->append(loc->d_names->type_names[weekday],
```

*Figure 1. Diagnosis Tool Architecture*
How does it work? Multiple Faulty Input Filtering

All of the filtering steps so far have been working on information gathered from single “bad” input run.

Taking the DBS and Dependence filtered candidate sets of ALL the “bad” input runs, any candidates not common between all sets is filtered out.

The resulting candidate set is reported to the user!
## Experiment

Tested on Squid, MySQL, Apache

Three Programs -> Eight Bugs

### Constraints

<table>
<thead>
<tr>
<th>Bug#</th>
<th>Application</th>
<th>LOC</th>
<th>Symptom</th>
<th>Bug Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug-1</td>
<td>Squid 2.3</td>
<td>70K</td>
<td>buffer overflow</td>
<td>Incorrect computation of buffer length leads to buffer overflow</td>
</tr>
<tr>
<td>Bug-2</td>
<td>MySQL 5.1.30</td>
<td>1019K</td>
<td>buffer overflow</td>
<td>Use of unsigned variable causes integer overflow which leads to buffer overflow</td>
</tr>
<tr>
<td>Bug-3</td>
<td>MySQL 5.1.30</td>
<td>1019K</td>
<td>Incorrect output</td>
<td>Wrong algorithm to convert microsecond field to integer results in incorrect value</td>
</tr>
<tr>
<td>Bug-4</td>
<td>MySQL 5.1.23a</td>
<td>1054K</td>
<td>buffer overflow</td>
<td>Use of a negative number along with an aggregate function results in seg fault</td>
</tr>
<tr>
<td>Bug-5</td>
<td>MySQL 5.0.18</td>
<td>949K</td>
<td>Incorrect output</td>
<td>Loss of precision in a sequence of computation results in wrong value</td>
</tr>
<tr>
<td>Bug-6</td>
<td>MySQL 5.0.18</td>
<td>949K</td>
<td>Incorrect output</td>
<td>Overflow during decimal multiplication results in garbage output</td>
</tr>
<tr>
<td>Bug-7</td>
<td>MySQL 5.0.15</td>
<td>937K</td>
<td>Incorrect output</td>
<td>Loss of data when inserting big values in a table</td>
</tr>
<tr>
<td>Bug-8</td>
<td>Apache 2.2</td>
<td>225K</td>
<td>buffer overflow</td>
<td>For particular value of output size, buffer overflow occurs</td>
</tr>
</tbody>
</table>
Difficulty In Diagnosis

<table>
<thead>
<tr>
<th>#Bug</th>
<th>Static #LOC executed in failed run</th>
<th>Distance (Dyn #LLVM inst)</th>
<th>Distance (Static #LOC)</th>
<th>Distance (Static #Functions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug-1</td>
<td>6927</td>
<td>12</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Bug-2</td>
<td>9822</td>
<td>18</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Bug-3</td>
<td>9982</td>
<td>86</td>
<td>27</td>
<td>10</td>
</tr>
<tr>
<td>Bug-4</td>
<td>11308</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Bug-5</td>
<td>7874</td>
<td>124</td>
<td>41</td>
<td>17</td>
</tr>
<tr>
<td>Bug-6</td>
<td>7835</td>
<td>114</td>
<td>36</td>
<td>17</td>
</tr>
<tr>
<td>Bug-7</td>
<td>9835</td>
<td>429</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>Bug-8</td>
<td>6217</td>
<td>32780</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>
## Results of Filtering False Positives

<table>
<thead>
<tr>
<th>Bug#</th>
<th>#Invs</th>
<th>Failed Invs</th>
<th>Slice</th>
<th>Dependence Filter</th>
<th>Multiple Faulty Inputs</th>
<th>Src-expr-tree</th>
<th>Root Cause in final step?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug-1</td>
<td>3358</td>
<td>357</td>
<td>30</td>
<td>9</td>
<td>9</td>
<td>49</td>
<td>Yes</td>
</tr>
<tr>
<td>Bug-2</td>
<td>5917</td>
<td>95</td>
<td>36</td>
<td>16</td>
<td>12</td>
<td>48</td>
<td>No</td>
</tr>
<tr>
<td>Bug-3</td>
<td>5942</td>
<td>93</td>
<td>27</td>
<td>9</td>
<td>6</td>
<td>64</td>
<td>Yes</td>
</tr>
<tr>
<td>Bug-4</td>
<td>6847</td>
<td>156</td>
<td>44</td>
<td>14</td>
<td>8</td>
<td>28</td>
<td>Yes</td>
</tr>
<tr>
<td>Bug-5</td>
<td>4566</td>
<td>130</td>
<td>34</td>
<td>18</td>
<td>17</td>
<td>89</td>
<td>Yes</td>
</tr>
<tr>
<td>Bug-6</td>
<td>4652</td>
<td>83</td>
<td>13</td>
<td>7</td>
<td>5</td>
<td>26</td>
<td>Yes</td>
</tr>
<tr>
<td>Bug-7</td>
<td>5836</td>
<td>153</td>
<td>35</td>
<td>17</td>
<td>11</td>
<td>152</td>
<td>Yes</td>
</tr>
<tr>
<td>Bug-8</td>
<td>2295</td>
<td>120</td>
<td>12</td>
<td>6</td>
<td>6</td>
<td>171</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Dynamic Backward Slicing pares 80% of input false positive candidates. Dependence Filter pares 58% of input false positive candidates. Multiple Faulty Input Filter effectively pares none.
Limitations

Limited to range invariants

System design may not be efficient if expanded to accommodate more invariants

Needs more robust input generation scheme
Conclusion

So what?
More robust algorithms
Less false positives
More bugs identified
Discussion

• The tool only considers loads, stores, and function returns for invariants (range invariants). It worked fine for the example bug, but how useful is this to developers in general?
Discussion

• Restriction to only range invariants was essential to the design of the system. Would expanding the invariants of interest be efficient?
Discussion

• The system is designed to primarily avoid missing possible candidates of root cause and secondarily reduce false positives. Would this still be a smart design choice if the invariant choices were expanded?
Discussion

• Is the experimental methodology sound?
Discussion

• The execution time for this tool was not reliable for the 8 test cases used in the authors’ experiment, ranging from 8 minutes to 4 hours depending on the program size and trace size. Would execution time scale well if more invariants were considered?