Effective Fault Localization Based on Minimum Debugging Frontier Set

Feng Li  Wei Huo  Congming Chen  Lujie Zhong  Xiaobing Feng

Institute of Computing Technology, Chinese Academy of Sciences

Zhiyuan Li

Purdue University
The Goal of automatic fault localization techniques
— locate an erroneous state (or a set of states) as close to the faulty point as possible.
Existing Approaches

Faulty program

Failing trace

Passing trace

Intended trace

A passing trace under a different program input ≠ The intended trace
Isolate a set of program states whose changes fix the failure
Tradeoff

- **Accuracy vs. Search space**

- $t$ is *the number of statement instances* in the failing run execution trace.
- $m$ is *the maximum number of mutation values* for any stmt instance in the failing trace.
Challenge

- how to reduce the search space and still achieve an acceptable degree of accuracy?

- Our solution: DelFal
  - allow state altering on MDFS only
  - employ a symbolic state altering and exploring strategy
DelFal Outline

- Motivation
- Key Ideas
- Overview of the Core Algorithm
- Evaluation
- Conclusion
Minimum Debugging Frontier Set (MDFS)

- Find the **min cut** of the analysis domain (in the form of *Dynamic Dependence Graph*)

Minimum debugging frontier set (MDFS)
Minimum Debugging Frontier Set (MDFS)

- Mimic the conventional divide-and-conquer debugging

Minimum debugging frontier set (MDFS)
Change states symbolically

- **Symbolically** alter and explore the state of each `stmt instance` in an MDFS
DelFal Overview

Input: point of failure, $pt$
expected states at $pt$ [Optional]

Preprocessing

Initial analysis domain
(DDG subgraph)

1. Extracting suspicious states

Stmt instances in MDFS

2. Identifying polluted entities

Polluted entities

3. Updating the analysis domain

Output: Fault Report (Root cause + Transitive causes)
DelFal Overview

Preprocessing
– extract the initial analysis domain

main () {
int x, y, z, m;
1 read("Enter 3 numbers: ", x, y, z);
2 m = x;
 /* fault-I: should be m = z */
3 if(y < z) /* if_1 */
4 if(x < y) /* if_2 */
5 m = y;
6 else if(x < z) /* if_3 */
7 m = y;
 /* fault-II: should be m = x */
8 else
9 if(x > y) /* if_4 */
10 m = y;
11 else if(x > z) /* if_5 */
12 m = x;
13 printf("Middle number is : ",m);
}

Input: \{x\leftarrow 2, y\leftarrow 1, z\leftarrow 3\}
Expected Output: m = 2
DelFal Overview

1. Extracting suspicious states
   - evaluate an MDFS for the current analysis domain

\[ \text{Entry} \]
\[ \text{read y read z} \]
\[ \text{tmp}_3 = x < z \]
\[ \text{v}_1 \]
\[ \text{v}_2 \]
\[ \text{v}_3 \]
\[ \text{Entry} \]
\[ \text{read x} \]
\[ \text{read y} \]
\[ \text{read z} \]

Subgraph DDG

\[ \text{Subgraph DDG}^1 \]
\[ \text{Entry} \]
\[ \text{read y} \]
\[ \text{read z} \]
\[ \text{if (tmp}_1 \) \]
\[ \text{if (tmp}_2 \) \]
\[ \text{else if (tmp}_3 \) \]
\[ m = y \]
\[ \text{tmp}_1 = y < z \]
\[ \text{tmp}_2 = x < y \]
\[ \text{tmp}_3 = x < z \]
\[ \text{v}_4 \]
\[ \text{v}_5 \]
\[ \text{v}_6 \]
\[ \text{v}_7 \]
\[ \text{v}_8 \]
\[ \text{v}_9 \]
\[ \text{v}_10 \]
\[ \text{v}_11 \]
\[ \text{v}_12 \]
DelFal Overview

2. Identifying polluted entities
   - symbolically modifying and exploring each suspicious state
2. Identifying polluted entities
   - symbolically modifying and exploring each suspicious state

Expected Output: \( m = 2 \)

Symbolic replacement: \( \gamma \) (sym\_y = 1)

Satisfiable
3. Updating the analysis domain – based on the result of step 2
In the given code snippet, the main function reads three numbers and determines the middle number based on certain conditions. The code contains several faults:

1. In line 3, `m = x; /* fault-I: should be m = z */` should actually be `m = z;` to correctly assign the middle value.
2. In line 7, `/* fault-II: should be m = x */` is misleading. The correct condition is `if (x < y) && (x < z)`, but the code does not handle cases where `x` is the middle value properly.
3. In line 8, `if (x > y) /* if_4 */` is incorrect. The correct condition for assigning `m` should be `if (x > y) && (x > z)`.
4. In line 10, `m = y; /* if_5 */` should be avoided as `m` is already assigned a value in line 9.

The root cause, as reported by DelFal, is the incorrect assignment of the middle number due to faulty conditionals. The diagram illustrates the control flow and data dependencies, highlighting the subgraph `DDG0` with the entry point `Entry` and the exit point `fake_Exit`. The termination condition is met when no more MDFS (Multiple Decision Flow Structures) exist in the current analysis domain.
Optimizations

- **(OPT1) checkpoint scheme**
- **(OPT2) refined MDFS**
  - Partition the failing trace into several intervals
  - Examine them in a reversed order
Optimizations

- **(OPT1) checkpoint scheme**
  - Partition the analysis domain into several intervals
  - Explore the intervals in a reverse order

- **(OPT2) refined MDFS**
  - An MDFS generated for a refined analysis domain
## Evaluation

<table>
<thead>
<tr>
<th>BugID</th>
<th>ProgVer</th>
<th>PatchDate</th>
<th>FaultDesc</th>
<th>TraceSize</th>
<th>SliceSize</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>grep-2.5.1</td>
<td>2004-07-01</td>
<td>missing assignment</td>
<td>9281</td>
<td>5597</td>
</tr>
<tr>
<td>2</td>
<td>grep-2.5.3</td>
<td>2006-05-19</td>
<td>missing initialization</td>
<td>8796</td>
<td>4395</td>
</tr>
<tr>
<td>3</td>
<td>grep-2.6</td>
<td>2010-03-25</td>
<td>incorrect goto</td>
<td>60753</td>
<td>31717</td>
</tr>
<tr>
<td>4</td>
<td>gdb-6.8</td>
<td>2008-12-19</td>
<td>design error</td>
<td>597756</td>
<td>106486</td>
</tr>
<tr>
<td>5</td>
<td>gdb-6.8</td>
<td>2008-12-31</td>
<td>design error</td>
<td>368330</td>
<td>37526</td>
</tr>
<tr>
<td>6</td>
<td>gdb-6.8.50</td>
<td>2009-09-22</td>
<td>design error</td>
<td>3538757</td>
<td>99408</td>
</tr>
<tr>
<td>7</td>
<td>gdb-7.0</td>
<td>2010-01-25</td>
<td>mutation operator</td>
<td>1354413</td>
<td>100014</td>
</tr>
<tr>
<td>8</td>
<td>sed-4.1</td>
<td>2005-05-16</td>
<td>incorrect guard</td>
<td>854</td>
<td>239</td>
</tr>
<tr>
<td>9</td>
<td>sed-4.1.5</td>
<td>2006-02-03</td>
<td>const error</td>
<td>3851</td>
<td>1376</td>
</tr>
<tr>
<td>10</td>
<td>sed-4.2a</td>
<td>2009-05-11</td>
<td>design error</td>
<td>3919</td>
<td>1439</td>
</tr>
<tr>
<td>11</td>
<td>tar-1.22</td>
<td>2009-07-30</td>
<td>missing function call</td>
<td>94807</td>
<td>21883</td>
</tr>
<tr>
<td>12</td>
<td>tar-1.22</td>
<td>2009-07-30</td>
<td>weak guard</td>
<td>74826</td>
<td>12169</td>
</tr>
<tr>
<td>13</td>
<td>tar-1.22</td>
<td>2009-10-07</td>
<td>design error</td>
<td>82800</td>
<td>12297</td>
</tr>
<tr>
<td>14</td>
<td>tar-1.23</td>
<td>2010-03-17</td>
<td>design error</td>
<td>107443</td>
<td>16370</td>
</tr>
<tr>
<td>15</td>
<td>tar-1.23</td>
<td>2010-06-28</td>
<td>call at wrong place</td>
<td>90586</td>
<td>17427</td>
</tr>
</tbody>
</table>
## Effectiveness

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>#States</th>
<th>Fault Report</th>
<th>Link to Fault</th>
<th>Where</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>dfa.c @ 1107</td>
<td>I_Clue [3]</td>
<td>D_Clue</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>grep.c @ 1897</td>
<td></td>
<td>D_Clue</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>dfasearch.c @ 301,388</td>
<td></td>
<td>D_Clue</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>valops.c @ 1926</td>
<td></td>
<td>D_Clue</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>infrun.c @ 3538</td>
<td></td>
<td>D_Clue</td>
</tr>
<tr>
<td>6</td>
<td>42</td>
<td>eval.c @ 1516, 1518</td>
<td></td>
<td>D_Clue</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>c-valprint.c @ 189</td>
<td></td>
<td>Hit</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>compile.c @ 479</td>
<td></td>
<td>D_Clue</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>compile.c @ 1277</td>
<td></td>
<td>D_Clue</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>compile.c @ 505</td>
<td></td>
<td>Hit</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>create.c @ 1437</td>
<td></td>
<td>D_Clue</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>create.c @ 1380</td>
<td>I_Clue [2]</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>33</td>
<td>create.c @ 1675-1687</td>
<td></td>
<td>Hit</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>unlink.c @ 104,105</td>
<td></td>
<td>Hit</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>list.c @ 94,126</td>
<td></td>
<td>D_Clue</td>
</tr>
</tbody>
</table>

DelFal precisely locates the faulty statements.

DelFal reports a transitive cause.

Direct effect of the faults are captured.
## Efficiency

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>Faulty Program</th>
<th>Size of Failing Trace</th>
<th>Time cost before &amp; after OPT (Unit: second)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tracing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>before</td>
</tr>
<tr>
<td>1</td>
<td>grep-2.5.1</td>
<td>9281</td>
<td>0.09</td>
</tr>
<tr>
<td>2</td>
<td>grep-2.5.3</td>
<td>8796</td>
<td>0.09</td>
</tr>
<tr>
<td>3</td>
<td>grep-2.6</td>
<td>60753</td>
<td>0.22</td>
</tr>
<tr>
<td>4</td>
<td>gdb-6.8</td>
<td>597756</td>
<td>0.79</td>
</tr>
<tr>
<td>5</td>
<td>gdb-6.8</td>
<td>368330</td>
<td>0.63</td>
</tr>
<tr>
<td>6</td>
<td>gdb-6.8.50</td>
<td>3538757</td>
<td>1.25</td>
</tr>
<tr>
<td>7</td>
<td>gdb-7.0</td>
<td>1354413</td>
<td>1.76</td>
</tr>
<tr>
<td>8</td>
<td>sed-4.1</td>
<td>854</td>
<td>0.03</td>
</tr>
<tr>
<td>9</td>
<td>sed-4.1.5</td>
<td>3851</td>
<td>0.17</td>
</tr>
<tr>
<td>10</td>
<td>sed-4.2a</td>
<td>3919</td>
<td>0.14</td>
</tr>
<tr>
<td>11</td>
<td>tar-1.22</td>
<td>94804</td>
<td>0.24</td>
</tr>
<tr>
<td>12</td>
<td>tar-1.22</td>
<td>74807</td>
<td>0.21</td>
</tr>
<tr>
<td>13</td>
<td>tar-1.22</td>
<td>82800</td>
<td>0.44</td>
</tr>
<tr>
<td>14</td>
<td>tar-1.23</td>
<td>107443</td>
<td>0.22</td>
</tr>
<tr>
<td>15</td>
<td>tar-1.23</td>
<td>90586</td>
<td>0.32</td>
</tr>
</tbody>
</table>

- **Total Time Cost**: About an hour
- **Additional Time Cost**: > 8 hours
Conclusion

- *DelFal* helps users effectively understand faults in real world programs in a reasonable time.

- *DelFal* reports either the faulty statements or their direct effect on the program behaviors for 13 out of 15 issues.
Thanks
BACKUP SLIDES
Why min-cut?

- Alternatives
  - Binary search
  - 1-step search

- MDFS
  - Avg. size: 6 (1 ~ 13)
  - Avg. slice size: 678 (27 ~ 2504)
Cost factors

- the number of iterations
- the average size of states to be explored per $\text{Iter}$
- the average size of code to be symbolically executed per $\text{Iter}$
/*----------------- Subtrace of Grep 2.6 ------------*/
grep (...) {
  ...
  437 buffer = xmalloc(bufalloc);
  438 bufbeg = buflim =
                ALIGN_TO(buffer+1, pagesize);
  1114 beg = bufbeg;
  ...
  // call EGexecute()
  ...
  301 len = patterns[i].regs.end[0] - start;
  ...
  // wrong jump,should be goto success;
  361 goto success_in_len;
  385 success:
  386 len = end - beg;
  387 success_in_len:
  388 match_size = len;
  ...
  // function return
  ...
  1019 *endp = b + match_size;
  1025 prtext (..., endp, ...);
  ...
  // produce the undesired output
  859 fwrite(beg, 1, lim - beg, stdout);
  ...
}
Limitations of DelFal

- Only deals with observable failures
  - Needs to know the failure site
  - The expected state at the site is *Optional, but helps substantially*

- Have limitations for faults associated with type
  - e.g. faults in declaration statements or those about numerical precisions

- State alteration is not a panacea
Multiple Faults
State-based Approach

Faulty program
Faulty Code

Failing trace

passing trace

Unavailable!

A passing trace under a different program input ≠

The intended trace
State-based Approach

Faulty program

Failing trace

Faulty Code

passing trace

Intended trace

Unavailable!

A passing trace under a different program input ≠ The intended trace