Finding and Understanding Bugs in C Compilers

Xuejun Yang, Yang Chen, Eric Eide, and John Regehr
University of Utah
PLDI 2011

Martin Velez
UC Davis ECS 289C
Tuesday, February 17, 2015
Motivation: “Compilers should be correct.”
Motivating Example

“Figure 1. We found a bug in the version of GCC that shipped with Ubuntu Linux 8.04.1 for x86. At all optimization levels it compiles this function to return 1; the correct result is 0. The Ubuntu compiler was heavily patched; the base version of GCC did not have this bug.”
Overview

CSmith
Randomized test-case generation tool

Bug Hunting
• 3 years
• 325 bug reports
• All compilers had bugs
  • crashed
  • generated wrong code
• Generates “random” C programs
• Covers a large subset of C
• Avoids undefined behaviors
• Avoids unspecified behaviors
Bug Hunting

$./a.out == $./a.out == $./a.out

True

False
Single Interpretation

- C99
  - 191 undefined behaviors
  - 52 unspecified behaviors

Sort of...
Design Goals

1. Well-formed and single meaning programs
2. Maximize Expressiveness
Included

- Function definitions
- Global and local variable definitions
- Control flow (if-else, function calls, for, return, break, continue, goto)
- Signed and unsigned integers
- Arithmetic, logical, and bitwise operations
- struct: nested and bit-width fields
- Arrays of and pointers to all supported types
- const and volatile

Excluded

- Strings
- Dynamic memory allocation
- floating-point types
- unions?
- recursion
- function pointers
“Grammar”

- PROGRAM ::= <type-def-list><var-def-list><func-def-list>

- func-def-list ::= func-def <func-def-list>

- func-def ::= type func-name { block }

- block ::= <declaration-list> <statement-list>

- statement-list ::= statement <statement-list>

- statement-list ::= statement <statement-list>

- statement ::= expr | control-flow | assignment | block

- control-flow ::= if ... | return | goto | for
Algorithm

Preliminary Step: struct declarations

Start at `main`

1. Select an allowable production from grammar
   - Consult probability table and then filter function
2. Select target (variable or function)
   - Dynamic probability table of potential targets
3. Select a type
   - Restricted (+, -, ...) or unrestricted
4. Nonterminal recursion
5. Update local environment with points-to facts
6. Safety checks
   - commit or rollback
int main(int argc, int* argv[]) {
    //call top_level function
    //compute checksum
    //print checksum
    int 0;
}

Board: Examples of Unsafe Code
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>use without initialization</td>
<td>explicit initializers, avoid jumping over initializers</td>
<td>—</td>
</tr>
<tr>
<td>qualifier mismatch</td>
<td>static analysis</td>
<td>—</td>
</tr>
<tr>
<td>infinite recursion</td>
<td>disallow recursion</td>
<td>—</td>
</tr>
<tr>
<td>signed integer overflow</td>
<td>bounded loop vars</td>
<td>safe math wrappers</td>
</tr>
<tr>
<td>OOB array access</td>
<td>bounded loop vars</td>
<td>force index in bounds</td>
</tr>
<tr>
<td>unspecified eval. order of function arguments</td>
<td>effect analysis</td>
<td>—</td>
</tr>
<tr>
<td>R/W and W/W conflicts</td>
<td>effect analysis</td>
<td>—</td>
</tr>
<tr>
<td>betw. sequence points</td>
<td>pointer analysis</td>
<td>—</td>
</tr>
<tr>
<td>access to out-of-scope stack variable</td>
<td>pointer analysis</td>
<td>—</td>
</tr>
<tr>
<td>null pointer dereference</td>
<td>pointer analysis</td>
<td>null pointer checks</td>
</tr>
</tbody>
</table>

**Safety Mechanisms**

Table 1. Summary of Csmith’s strategies for avoiding undefined and unspecified behaviors. When both a code-generation-time and code-execution-time solution are listed, Csmith uses both.
Design Trade-offs

• Allow implementation-defined behavior
• No ground truth
• No guarantee of termination
• Target middle-end

10% of Smith generated programs are non-terminating
Results
1) Opportunistic Bug Finding

- 3 years
- Focus on GCC and LLVM
- Commercial Compilers (Not receptive)
- Compcert (front-end bugs only)

<table>
<thead>
<tr>
<th></th>
<th>GCC</th>
<th>LLVM</th>
<th>Others (CIL, TCC, Open64, Compcert, commercial)</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>25</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Total</td>
<td>79</td>
<td>202</td>
<td>44</td>
<td>325</td>
</tr>
</tbody>
</table>
2) Comparison across different compiler versions

1M CSmith programs
2) Comparison across different compiler versions

1M CSmith programs
–O0, –O1, –O2, –Os, and –O3
3) # of Bugs v.s. Test-case Size

Figure 4. Number of distinct crash errors found in 24 hours of testing with Csmith-generated programs in a given size range.
4) Other tools

Figure 5. Comparison of the ability of five random program generators to find distinct crash errors
Table 3. Augmenting the GCC and LLVM test suites with 10,000 randomly generated programs did not improve code coverage much.

<table>
<thead>
<tr>
<th></th>
<th>Line Coverage</th>
<th>Function Coverage</th>
<th>Branch Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>make check-c</td>
<td>75.13%</td>
<td>82.23%</td>
</tr>
<tr>
<td></td>
<td>make check-c &amp; random</td>
<td>75.58%</td>
<td>82.41%</td>
</tr>
<tr>
<td>% change</td>
<td>+0.45%</td>
<td>+0.13%</td>
<td>+0.85%</td>
</tr>
<tr>
<td>absolute change</td>
<td>+1,482</td>
<td>+33</td>
<td>+4,471</td>
</tr>
<tr>
<td>Clang</td>
<td>make test</td>
<td>74.54%</td>
<td>72.90%</td>
</tr>
<tr>
<td></td>
<td>make test &amp; random</td>
<td>74.69%</td>
<td>72.95%</td>
</tr>
<tr>
<td>% change</td>
<td>+0.15%</td>
<td>+0.05%</td>
<td>+0.26%</td>
</tr>
<tr>
<td>absolute change</td>
<td>+655</td>
<td>+74</td>
<td>+926</td>
</tr>
</tbody>
</table>

5) Code Coverage

Table 3. Augmenting the GCC and LLVM test suites with 10,000 randomly generated programs did not improve code coverage much.
Where are the Bugs?

<table>
<thead>
<tr>
<th></th>
<th>GCC</th>
<th>LLVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front end</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Middle end</td>
<td>49</td>
<td>75</td>
</tr>
<tr>
<td>Back end</td>
<td>17</td>
<td>74</td>
</tr>
<tr>
<td>Unclassified</td>
<td>13</td>
<td>43</td>
</tr>
<tr>
<td>Total</td>
<td>79</td>
<td>202</td>
</tr>
</tbody>
</table>

**Table 4.** Distribution of bugs across compiler stages. A bug is unclassified either because it has not yet been fixed or the developer who fixed the bug did not indicate what files were changed.
Board: Examples of Wrong-Code Bugs

```analysis
if(safety_check) {
  transformation
}
```
Demo: CSmith
That's all Folks!