CS 526

Advanced Compiler Construction

http://misailo.cs.Illinois.edu/courses/cs526

TESTING COMPILERS

Today's Topic

The compilers are software written by humans, and thus can have errors.

The subtle errors in compilers are especially critical as the compiled programs would behave strangely without a good reason?

How do we find errors or make sure they are absent?

How common are these bugs?

LLVM

Tue Mar 12 2019 11:12:49 PDT
Resolved all aliases in my C code

Hide Search Description

Status: UNCONFIRMED, NEW, CONFIRMED, REOPENED **Content:** bug compiler

This result was limited to 500 bugs. See all search results for this query.

ID	Product	Comp	Assignee	Status	Resolution	Summary	Changed
34376	Build sc	cmake	unassignedbugs	NEW	---	Trying to find __atomic_fetch_add_4 fails on MacOSX	2017-08-30
35935	clang	C++	unassignedclangbugs	NEW	---	tooling::ToolInvocation catastrophic failure if not -fsyntax-only	2018-01-12
18276	clang	LLVM Cod	unassignedclangbugs	NEW	---	clang breaks OpenSSL library when compiling with -O2 or -O3	2014-01-06
26463	clang	C++	unassignedclangbugs	NEW	---	Run-time assert while building Chromium	2016-02-03
39108	clang	C++	unassignedclangbugs	NEW	---	constexpr std::string_view global no longer compiles	2018-09-28
35160	tools	lto	unassignedbugs	NEW	---	llvm-dsymutil fails with 'LLVM ERROR: inconsistency in registered CommandLine options' when built with LLVM_LINK_LLVM_DYLIB=ON	2017-11-01
34770	clang	C++11	unassignedclangbugs	NEW	---	Android Native Compile release fails	2017-09-28
31070	clang	C++11	unassignedclangbugs	NEW	---	clang 3.9 crashes on valid C++11 code on VS 2015, Intel 17 and gcc 4.8	2016-11-19
20738	clang	OpenCL	unassignedclangbugs	NEW	---	Radeon: code generation for GCN GPUs seems to be broken. Clpeak benchmark outputs many errors ("Can't spill VGPR!") and then GPU hangs.	2016-04-21
37903	clang	C++'17	unassignedclangbugs	NEW	---	Clang/LLVM 7.0.1 on Windows initializes inline static data member multiple times	Thu 12:17
26389	clang	LLVM Cod	unassignedclangbugs	REOP	---	[x86-64] clang generate wrong instruction for cygwin	2016-02-03
23783	Build sc	cmake	unassignedbugs	NEW	---	Unable to run "check-all" target when building LLVM 3.6.1	2015-06-08
33435	clang	C++11	unassignedclangbugs	NEW	---	template template argument bad deduction	2017-08-10
35100	clang	Formatte	unassignedclangbugs	NEW	---	ClangFormat extension does not work in Visual Studio 2013	2017-10-26
40680	clang	-New Bug	unassignedclangbugs	NEW	---	clang: error: unable to execute command: Segmentation fault when compiling Apache Qpid Dispatch	2019-02-10
20326	clang	Headers	unassignedclangbugs	NEW	---	recent header breakage in clang	2014-07-21
39622	clang	-New Bug	unassignedclangbugs	NEW	---	Clang report header file not found when using -E but file exists	2018-11-11
29156	new-bugs	new bugs	unassignedbugs	NEW	---	reproducible assert with -Wdocumentation with clang r279749 with creduce'd test case	2017-10-03
34212	librarie	Backend:	unassignedbugs	NEW	---	r4 writes inserted overwriting a register that is supposed to be preserved across function calls	2017-11-14
30910	librarie	MCJIT	unassignedbugs	NEW	---	Deleting MCJIT execution engine causes SIGSEGV, Segmentation fault in libstdc++ exceptions handling (__cxa_throw)	2016-11-05

How common are these bugs?

GCC

Tue Mar 12 2019 18:16:17 UTC

"And they all lived happily ever after" and "Let's do infinite sequels" are somewhat incompatible concepts.

This list is too long for Bugzilla's little mind; the Next/Prev/First/Last buttons won't appear on individual bugs.

Hide Search Description

Resolution: --- **Component:** c++ **Product:** gcc

2906 bugs found.

ID	Product	Comp	Assignee ▲	Status ▲	Resolution	Summary	Changed ▲
24591	gcc	c++	unassigned	NEW	---	poor diagnostic with a missing { in a class definition	2005-10-31
22431	gcc	c++	unassigned	NEW	---	-Weff++ warns about missing usage of const initalizer list in synthesized ctors	2005-12-25
21139	gcc	c++	unassigned	NEW	---	Improve handling of function-scope statics on platforms without weak symbols	2005-12-30
18638	gcc	c++	unassigned	NEW	---	macros should be expanded in #pragma align for C++	2006-02-02
24847	gcc	c++	unassigned	NEW	---	Instantiates un-called copy constructor	2006-02-26
25322	gcc	c++	unassigned	NEW	---	ISO compliance of defining structs in anonymous unions	2006-05-11
18069	gcc	c++	unassigned	NEW	---	Contradicting type and variable attributes	2006-06-04
19502	gcc	c++	unassigned	NEW	---	duplicate diagnostic for invalid template constant parameter	2006-09-03
19965	gcc	c++	unassigned	NEW	---	Invalid member declaration diagnosed late	2006-09-03
29556	gcc	c++	unassigned	NEW	---	Expect error when 'using namespace std' is declared when std namespace is not defined	2006-10-23
30060	gcc	c++	unassigned	NEW	---	Error/warning on invalid code (duplicate identifier for enum/class) should be more specific	2007-01-20
20040	gcc	c++	unassigned	NEW	---	A new expression must check the access level of delete operator	2007-01-22
23263	gcc	c++	unassigned	NEW	---	Incomprehensible message for invalid attempt to partially specialize a member	2007-03-12
31326	gcc	c++	unassigned	NEW	---	data members in multiple inheritance	2007-03-23
17000	gcc	c++	unassigned	NEW	---	parse error calling member template function of non-lvalue from within template class member	2007-04-20
32143	gcc	c++	unassigned	UNCO	---	decl rtl generated with incorrect visibility	2007-05-29
19501	gcc	c++	unassigned	NEW	---	Redundant "template" keyword rejected	2007-06-10
31164	gcc	c++	unassigned	UNCO	---	Problem with GCC 4.1 and Boost signals	2007-07-01
15269	gcc	c++	unassigned	NEW	---	__attribute__((deprecated)) broken with inline, ignored with pure virtual, misreported after definition	2007-09-23

How do the programs fail?

The compiler crashes, or gives an assertion failure

(I bet you've seen those in your MP1)

How do the programs fail?

PHP version 5.3.4, the interpreter enters an infinite loop:

```
<?php
    $result = 0// ...
    $result = $result + 2.2250738585072011e-308;
    printf( "Final result: %.17e \n", $result);
?>
```

Similar thing happened with Java VM…

How do the programs fail?

The code may be generated just fine, but the optimization introduced an error or assumed something that it shouldn't have

```
int foo (void) {
  signed char x = 1;
  unsigned char y = 255;
  return x > y;
}
```

Bug found by CSmith (PLDI'11) in the version of GCC that shipped with Ubuntu 8.04. 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

Survey of some potential issues with compiler optimizations:

- Dangerous Optimizations and the Loss of Causality
- https://pubweb.eng.utah.edu/~cs5785/slides-f10/Dangerous+Optimizations.pdf

Compiler Fuzzing

Key idea: Generate many programs and compile them. The compiler should still be able to produce (ideally correct) code for these programs

Questions:

- How to generate programs?
- How to know they are correct?
- How to identify where the error may be?

CSmith

Generates arbitrary C programs that conform to the C99 standard.

- *Finding and Understanding Bugs in C Compilers Xuejun Yang, Yang Chen, Eric Eide, John Regehr (PLDI '11)*
- Explores atypical combinations of C language features
- Found many bugs in existing compilers
- Key challenge is targeting program generation to more likely reveal potential problems
- Trivia: the program size that helped discover most bugs was around 82KB

Fuzzing (for various purposes) is a vibrant research area these days

Checking for Correctness

1. Make sure the compiler is not behaving unexpectedly: crashing, diverging, etc.
2. Compare generated programs:
   - Compile with multiple compilers/versions or optimization levels and see if the results differ (see e.g., CSmith)
   - Change a program in some controlled manner (V. Le at al. PLDI'14, OOPSLA'15) http://web.cs.ucdavis.edu/~su/emi-project/

Example from EMI (PLDI'14)

```
struct tiny { char c; char d; char e; };
void foo(struct tiny x) {
  if (x.c != 1) abort();
  if (x.e != 1) abort();
}
int main() {
  struct tiny s;
  s.c = 1; s.d = 1; s.e = 1;
  foo(s);
  return 0;
}
```

A bug in the LLVM optimizer causes this miscompilation. The developers believe that the Global Value Numbering (GVN) optimization turns the struct initialization into a single 32-bit load. Subsequently, the Scalar Replacement of Aggregates (SRoA) optimization decides that the 32-bit load is undefined behavior, as it reads past the end of the struct, and thus does not emit the correct instructions to initialize the struct. The developer who fixed the issue characterized it as

> *"... very, very concerning when I got to the root cause, and very annoying to fix."*

Figure 2: Reduced version of the code in Figure 1b for bug reporting. (`http://llvm.org/bugs/show_bug.cgi?id=14972`)

```
int a, b, c, d, e;
int main() {
  for (b = 4; b > -30; b--)
    for (; c;)
      for (;;) {
        e = a > 2147483647 - b;
        if (d) break;
      }
  return 0;
}
```

Partial Redundancy Elimination (PRE) detects the expression "`e2147483647 - b`" as loop invariant. Loop Invariant Motion (LIM) tries to move it up from the innermost loop to the body of the outermost loop. Unfortunately, this optimization is problematic, as GCC then detects a signed overflow in the program's optimized version and this (incorrect) belief of the existence of undefined behavior causes the compiler to generate non-terminating code (and the bogus warning at `-O2`).

Figure 3: GCC miscompiles this program to an infinite loop instead of immediately terminating with no output. (`http://gcc.gnu.org/bugzilla/show_bug.cgi?id=58731`)

Test Oracles

1. Detect crashing or hanging compilers

2. Differential Testing

- Try multiple compilers – do they result in the same outcome
- Cross-compiler, Cross-optimization, Cross-version
- Compare the results of the compiled program or the code itself

3. Metamorphic testing

- Change the input program in a way that you expect the output to change.
- E.g., if the program compiles f(x) { return x } then it should compile g(x) { return 2*x }
- But more often, these relations are equivalences, even for a particular input, e.g., f(x) { return x + 2 } and g() { return 4 } are equivalent when testing f(2) and g().

Example Metamorphic Relations

Paper	Metamorphic relation	How to construct metamorphic relations
Tao et al. [114]	Equivalence relation	Constructing equivalent expressions, assignment blocks, and submodules
Le et al. [72]	Equivalence relation under a given set of test inputs	Deleting code in the dead regions under the set of test inputs
Le et al. [73]	Equivalence relation under a given set of test inputs	Deleting and inserting code in the dead regions under the set of test inputs
Sun et al. [111]	Equivalence relation under a given set of test inputs	Inserting code in both the live and dead regions by synthesizing valid semantic-preserving code snippets under the set of test inputs
Donaldson and Lascu [43]	Equivalence relation	Injecting dead code into test programs
Nakamura and Ishiura [89]	Equivalence relation	Applying a set of equivalent transformation rules on test programs
Donaldson et al. [42]	Equivalence relation	Applying a set of (essentially) semantics-preserving transformations on high-value graphics shaders
Samet [99–101]	Equivalence relation	Converting a source program and the object program into an intermediate representation, respectively

Table from A Survey of Compiler Testing, CSUR February 2020

Program Reduction

Aid in fault localization by reducing the size of the program that reveals an error.

- Once we know the error, we can start removing program statements for as long as the smaller program still reveals the same error
- See e.g.,
  - CReduce, PLDI'12
  - Perses, ICSE'18

Verification

- Verified compilation: **CompCert** is a verified, optimizing compiler for a large subset of C
  - http://compcert.inria.fr/compcert-C.html
  - Backed by powerful solvers
  - The executable code ***is formally proved*** to behave exactly as specified by the program semantics

- Formalizing semantics: An Executable Formal Semantics of C with Applications; Chucky Ellison and Grigore Rosu (POPL'12)
  - http://www.kframework.org

Performance Measuring

Not trivial! There are more concerns than just run and collect timings

Memory layouts can have up to 10% execution time variability:

- Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter Sweeney, "Producing Wrong Data Without Doing Anything Obviously Wrong!" (ASPLOS'09)

One solution: Stabilizer (ASPLOS'13) for performance measurement with randomized memory layouts:

- https://emeryberger.com/research/stabilizer/

Guidelines for measuring time: Kalibera and Jones, "Rigorous Benchmarking in Reasonable Time" (ISMM'13)

Identify Missed Optimizations

Compare the generated code statically / dynamically

- Gergo Barany. 2018. Finding missed compiler optimizations by differential testing. *International Conference on Compiler Construction*.
- Atsushi Hashimoto and Nagisa Ishiura. 2016. Detecting arithmetic optimization opportunities for C compilers by randomly generated equivalent programs. *IPSJ Trans. Syst. LSI Des. Methodol.* 9 (2016),
- Tipp Moseley, Dirk Grunwald, and Ramesh Peri. 2009. OptiScope: Performance accountability for optimizing compilers. *International Symposium on Code Generation and Optimization*.

For more details, see

A Survey of Compiler Testing, CSUR February 2020 ()
https://dl.acm.org/doi/10.1145/3363562

Csmith: https://embed.cs.utah.edu/csmith/
Creduce: https://embed.cs.utah.edu/creduce/
EMI: https://web.cs.ucdavis.edu/~su/emi-project/