

Automatic program generation for
detecting vulnerabilities and errors
in compilers and interpreters

0368-3500

Nurit Dor

Shir Landau-Feibish

Noam Rinetzky

Preliminaries

- Students will group in teams of 2-3 students.
- Each group will do one of the projects presented.

Administration

- Workshop meetings will take place only on Thursdays 12-14
 - No meetings (with us) during other hours
- Attendance in all meetings is mandatory
- Grading: 100% of grade will be given after final project submission.
- Projects will be graded based on:
 - Code correctness and functionality
 - Original and innovative ideas
 - Level of technical difficulty of solution

Administration

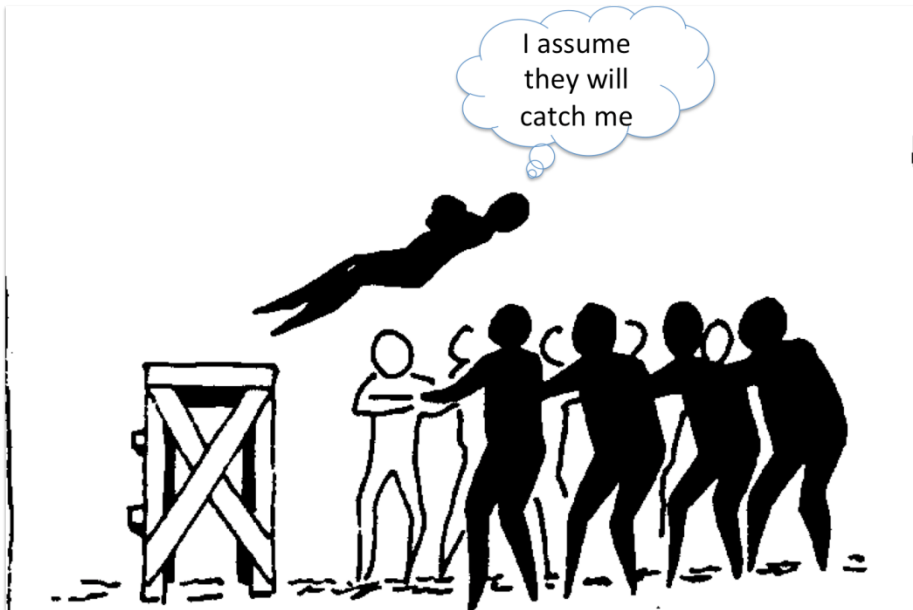
- Workshop staff should be contacted by email.
- Please address all emails to all of the staff:
 - Noam Rinetzky - maon@cs.tau.ac.il
 - Nurit Dor - nurit.dor@gmail.com
 - Shir Landau Feibish – lfshir@gmail.com
- Follow updates on the workshop website:
<http://www.cs.tau.ac.il/~maon/teaching/2014-2015/workshop/workshop1415b.html>

Tentative Schedule

- Meeting 1, 11/03/2015 (today)
 - Project presentation
- Meeting 2, 16/04/2015
 - Each group presents its initial design
- Meeting 3, 14/05/2015
 - Progress report – meeting with each group
- Meeting 4, 18/06/2015
 - First phase submission
- Submission: 01/09/2015
- Presentation: ~08/09/2015
 - Each group separately

Automatic program generation for
detecting vulnerabilities and errors
in compilers and interpreters

Programming Errors



“As soon as we started programming, we found to our surprise that it wasn’t as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.”

—Maurice Wilkes,
Inventor of the EDSAC,
1949

Compiler bugs?

- Most programmers treat compiler as a 100% correct program
- Why?
 - Never found a bug in a compiler
 - Even if they do, they don't understand it and solve the problem by “voodoo programming”
 - A compiler is indeed rather thoroughly tested
 - Tens of thousands of testcases
 - Used daily by so many users

Small Example

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

- Bug in GCC for Ubuntu compiles this function to return 1

FUZZERS

What is Fuzzing?

- Fuzzing is a testing approach
 - Test cases generated by a program.
 - Software under test is activated on those testcases
 - Monitored at run-time for failures

Naïve Fuzzing

- Miller et al 1990
- Send “random” data to application.
 - Long printable and non-printable characters with and without null byte
- 25-33% of utility programs (emacs, ftp,...) in unix crashed or hanged



Naïve Fuzzing

- Advantages:
 - Amazingly simple
- Disadvantage:
 - inefficient
 - Input often requires structures
 - random inputs are likely to be rejected
 - Inputs that would trigger a crash is a very small fraction, probability of getting lucky may be very low
 - Today's security awareness is much higher

Mutation Based Fuzzing

- Little or no knowledge of the structure of the inputs is assumed
- Anomalies are added to existing valid inputs
- Anomalies may be completely random or follow some heuristics
- Requires little to no set up time
- Dependent on the inputs being modified
- May fail for protocols with checksums, those which depend on challenge response, etc.

Mutation Based Example: PDF Fuzzing

- Google .pdf (lots of results)
- Crawl the results and download lots of PDFs
- Use a mutation fuzzer:
 1. Grab the PDF file
 2. Mutate the file
 3. Send the file to the PDF viewer
 4. Record if it crashed (and the input that crashed it)

Generation Based Fuzzing

- Test cases are generated from some description of the format: RFC, documentation, etc.
- Anomalies are added to each possible spot in the inputs
- Knowledge of protocol should give better results than random fuzzing
- Can take significant time to set up

Example Specification for ZIP file

```
1 <!-- A. Local file header -->
2 <Block name="LocalFileHeader">
3   <String name="lfh_Signature" valueType="hex" value="504b0304" token="true" mut
4   <Number name="lfh_Ver" size="16" endian="little" signed="false"/>
5   ...
6   [truncated for space]
7   ...
8   <Number name="lfh_CompSize" size="32" endian="little" signed="false">
9     <Relation type="size" of="lfh_CompData"/>
10  </Number>
11  <Number name="lfh_DecompSize" size="32" endian="little" signed="false"/>
12  <Number name="lfh_FileNameLen" size="16" endian="little" signed="false">
13    <Relation type="size" of="lfh_FileName"/>
14  </Number>
15  <Number name="lfh_ExtraFldLen" size="16" endian="little" signed="false">
16    <Relation type="size" of="lfh_FldName"/>
17  </Number>
18  <String name="lfh_FileName"/>
19  <String name="lfh_FldName"/>
20  <!-- B. File data -->
21  <Blob name="lfh_CompData"/>
22 </Block>
```

Mutation vs Generation

Mutation Based

Easy to implement, no need to understand the input structure

General implementation

Effectiveness is limited by the initial testcases

Coverage is usually not improved

Generation based

Can be labor intensive to implement especially for complex input (file formats)

Implementation for specific input

Can produce new testcases

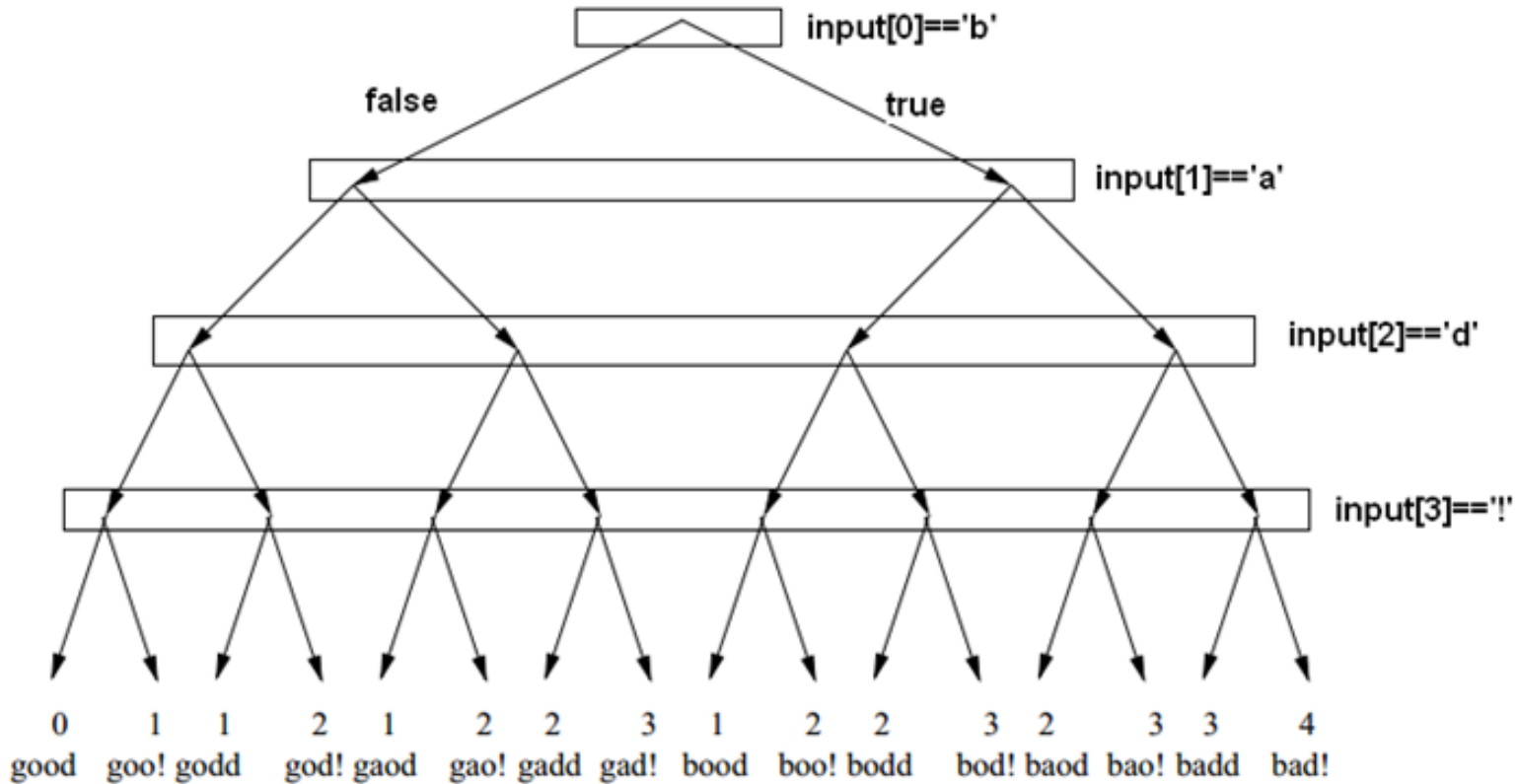
Coverage is usually improved

Constraint Based Fuzzing

- Mutation and generation based fuzzing will probably not reach the crash

```
void test(char *buf)
{
    int n=0;
    if(buf[0] == 'b') n++;
    if(buf[1] == 'a') n++;
    if(buf[2] == 'd') n++;
    if(buf[3] == '!') n++;
    if(n==4) {
        crash();
    }
}
```

Constraint Based Fuzzing



CSMITH

Csmith

- From the University of Utah
- Csmith is a tool that can generate random C programs
 - Only valid C99 standard



Random Generator: Csmith



C program



...

results

vote

majority

minority



```

#include "csmith.h"
static long __undefined;

/* --- Struct/Union Declarations --- */
/* --- GLOBAL VARIABLES --- */
static int32_t g_7 = 9L;
static int32_t *g_6 = &g_7;
static int64_t g_10 = (-1L);
static const int32_t **const volatile g_14[6][5][6] = {{{{0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}},
static volatile int32_t g_29 = (-6L); /* VOLATILE GLOBAL g_29 */
static int32_t g_30 = 0xDE63B691L;
static uint64_t g_43 = 0xA8FD5F396434D12ELL;

```

```

    int64_t l_464 = (-1L);
lbl_474:
    if ((g_169 & (safe_mul_func_uint8_t_u_u(g_43, (l_431 != 0))))))
    { /* block id: 412 */
        const int32_t *l_432 = 0;
        int32_t *l_434 = &g_30;
        (*l_434) ^= (*g_105);
        for (g_7 = 0; (g_7 <= (-17)); g_7 = safe_sub_func_uint64_t_u_u(g_7, 1))
        { /* block id: 416 */
            const int16_t l_441 = 7L;
            (*l_434) = (safe_add_func_int64_t_s_s((((~(g_30 == (safe_mul_func_uin
            return g_232;
        }
        (*l_434) = (0U | (((safe_mod_func_int32_t_s_s((*l_434), (*l_417))) ^ g_133)
        l_451 = (safe_add_func_int32_t_s_s((*l_434), (safe_mul_func_uint8_t_u_u
    }

```

```

int main (int argc, char* argv[])
{
    int i, j, k;
    int print_hash_value = 0;
    if (argc == 2 && strcmp(argv[1], "1") == 0) print_hash_value = 1;
    platform_main_begin();
    crc32_gentab();
    func_1();
    transparent_crc(g_7, "g_7", print_hash_value);
    transparent_crc(g_10, "g_10", print_hash_value);
    transparent_crc(g_29, "g_29", print_hash_value);

```



```
jxyang@gamow: ~/csmith/demo$ ls
jxyang@gamow: ~/csmith/demo$ ../src/csmith --bitfields --packed-struct -s 4276401180 > test.c
jxyang@gamow: ~/csmith/demo$ ls -l
total 56
-rw-r--r-- 1 jxyang embed 52378 2011-06-07 08:49 test.c
jxyang@gamow: ~/csmith/demo$ regehr/z/bin/current-gcc -v
Using built-in specs.
COLLECT_GCC=/home/regehr/z/bin/current-gcc
COLLECT_LTO_WRAPPER=/uusoc/exports/scratch/regehr/z/compiler-install/gcc-r174655-install/bin/./libexec/gcc/x86_64-unknown-linux-gnu/4.7.0/lto-wrapper
Target: x86_64-unknown-linux-gnu
Configured with: ../configure --with-libelf=/usr/local --enable-lto --prefix=/home/regehr/z/compiler-install/gcc-r174655-install --program-prefix=r174655- --enable-languages=c,c++
Thread model: posix
gcc version 4.7.0 20110605 (experimental) (GCC)
jxyang@gamow: ~/csmith/demo$ regehr/z/bin/current-gcc -O3 -I../runtime test.c
test.c: In function 'func_1':
test.c:239:27: warning: comparison of distinct pointer types lacks a cast [enabled by default]
test.c: In function 'func_25':
test.c:625:48: warning: comparison of distinct pointer types lacks a cast [enabled by default]
test.c:661:133: warning: comparison of distinct pointer types lacks a cast [enabled by default]
test.c:679:287: warning: comparison of distinct pointer types lacks a cast [enabled by default]
test.c: In function 'main':
test.c:1055:5: internal compiler error: in vect_enhance_data_refs_alignment, at tree-vect-data-refs.c:1551
Please submit a full bug report,
with preprocessed source if appropriate.
See <http://gcc.gnu.org/bugs.html> for instructions.
```

Why Csmith Works

- **Unambiguous:** avoid undefined or unspecified behaviors that create ambiguous meanings of a program

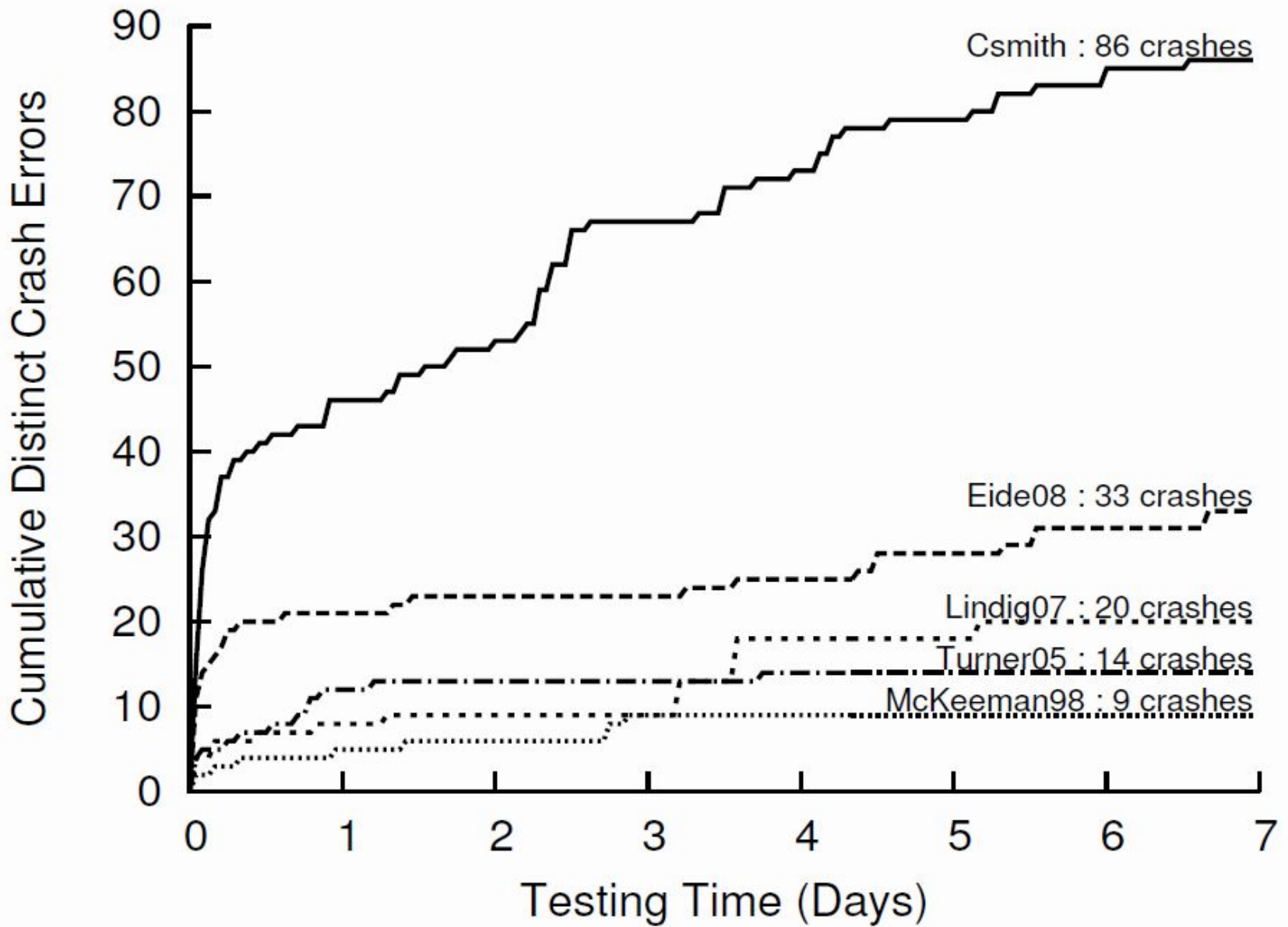
Integer undefined behavior
Use without initialization
Unspecified evaluation order

Use of dangling pointer
Null pointer dereference
OOB array access

- **Expressiveness:** support most commonly used C features

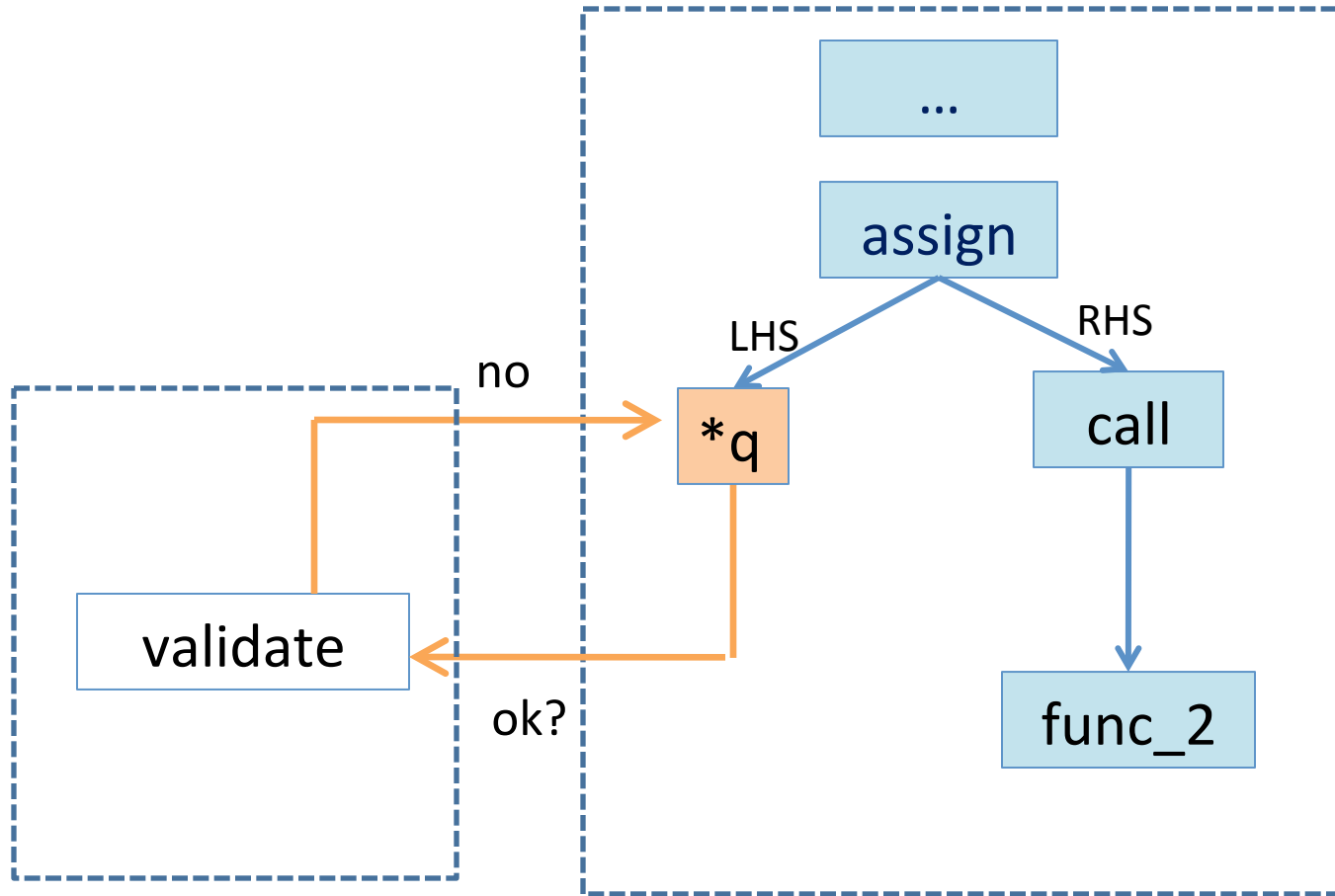
Integer operations
Loops (with break/continue)
Conditionals
Function calls

Const and volatile
Structs and Bitfields
Pointers and arrays
Goto



Avoiding Undefined/unspecified Behaviors

Problem	Generation Time Solution	Run Time Solution
Integer undefined behaviors	<ul style="list-style-type: none">• Constant folding/propagation• Algebraic simplification	Safe math wrappers
Use without initialization	explicit initializers	
OOB array access	Force index within range	Take modulus
Null pointer dereference	Inter-procedural points-to analysis	
Use of dangling pointers	Inter-procedural points-to analysis	
Unspecified evaluation order	Inter-procedural effect analysis	

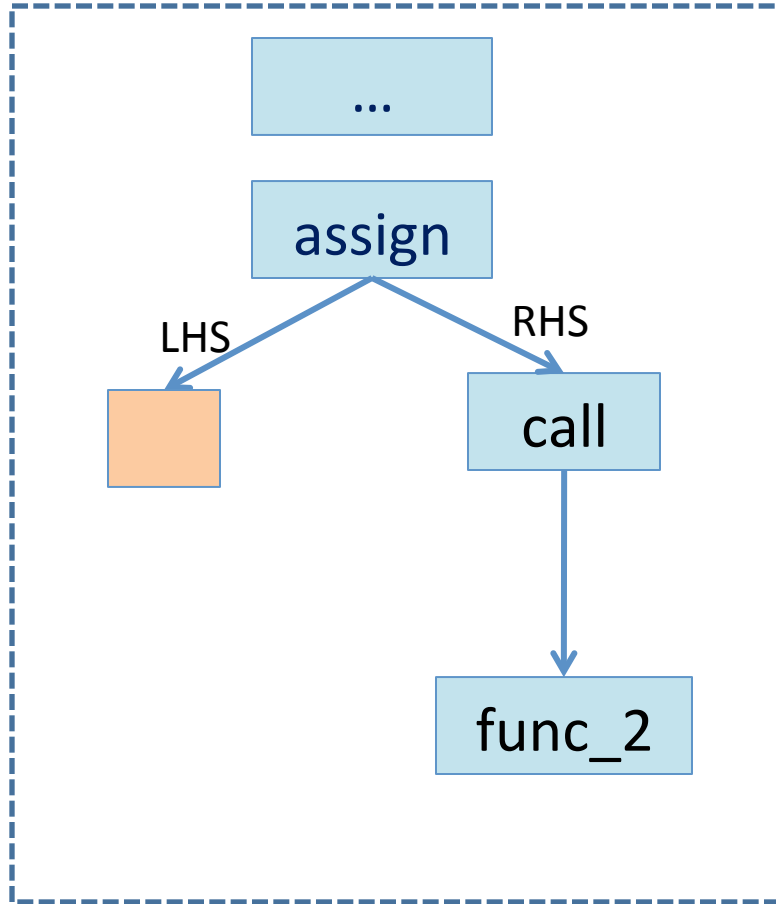


Generation Time Analyzer

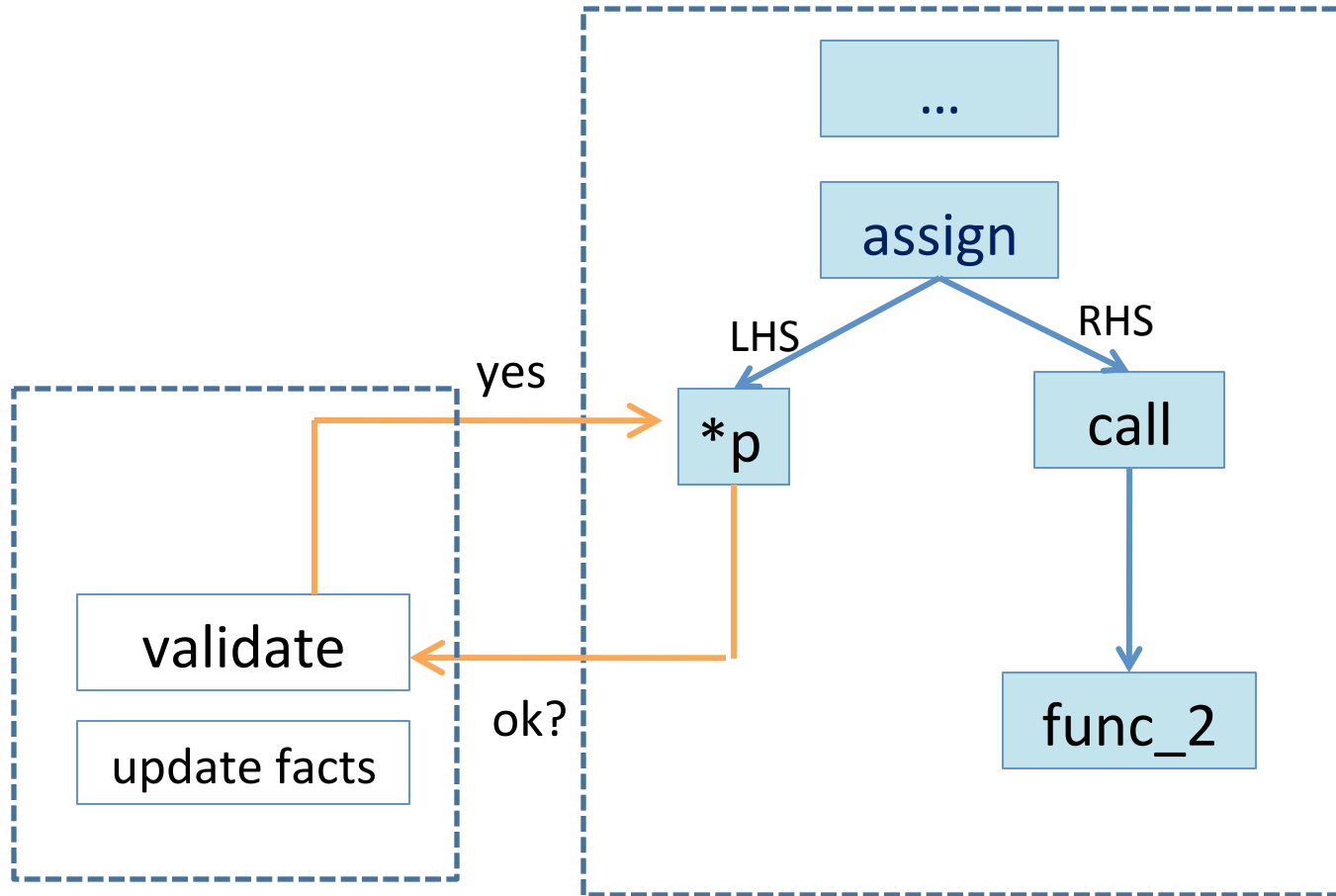
Code Generator



Generation Time Analyzer



Code Generator



Generation Time Analyzer

Code Generator

- From March, 2008 to present:

Compiler	Bugs reported (fixed)
GCC	104 (86)
LLVM	228 (221)
Others (Compcert, icc, armcc, tcc, cil, suncc, open64, etc)	50
Total	382

Accounts for 1% total valid GCC bugs reported in the same period

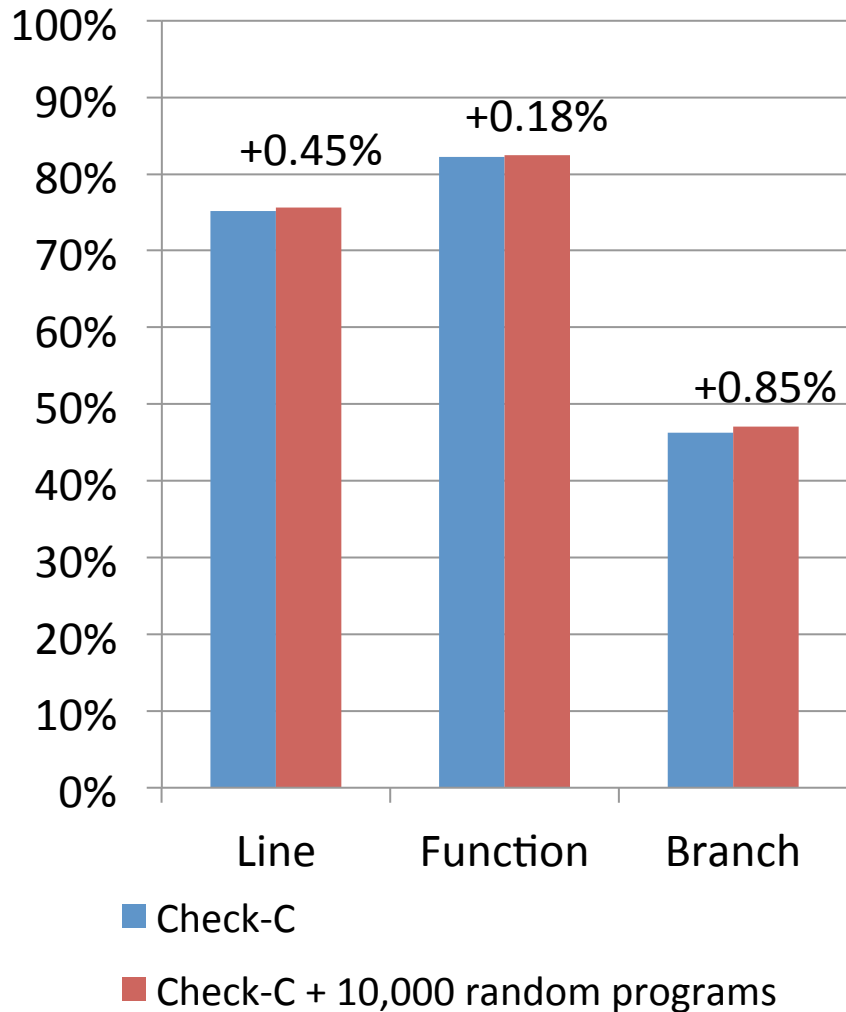
Accounts for 3.5% total valid LLVM bugs reported in the same period

- Do they matter?
 - 25 priority 1 bugs for GCC
 - 8 of reported bugs were re-reported by others

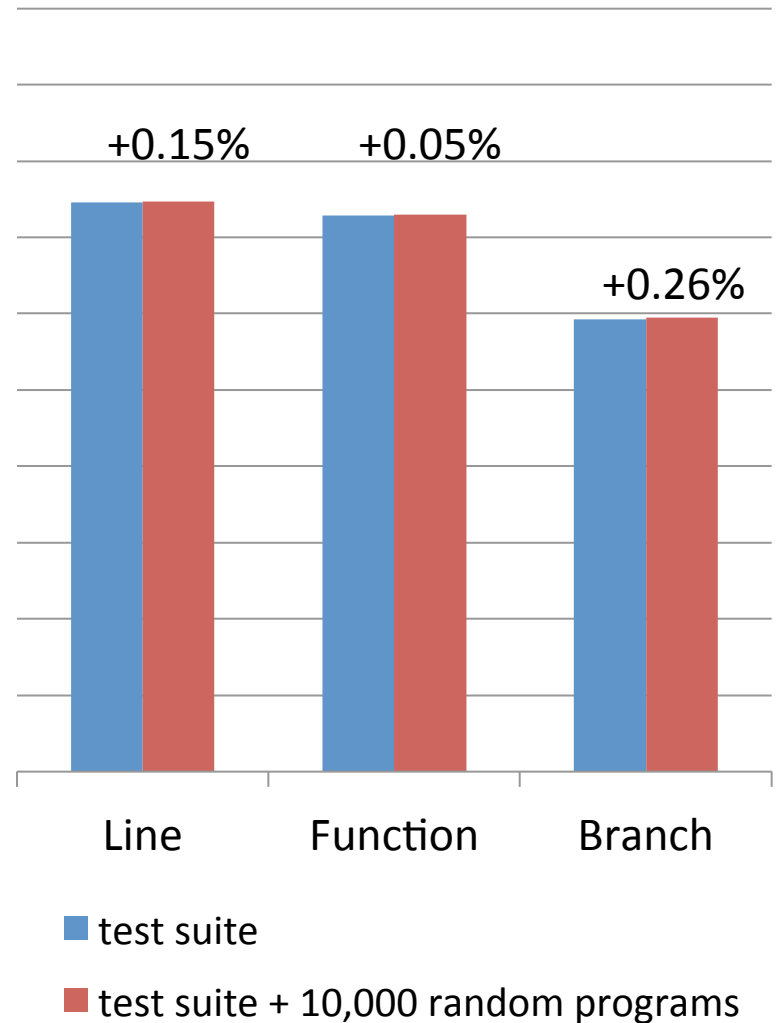
Bug Dist. Across Compiler Stages

	GCC	LLVM
Front end	1	11
Middle end	71	93
Back end	28	78
Unclassified	4	46
Total	104	228

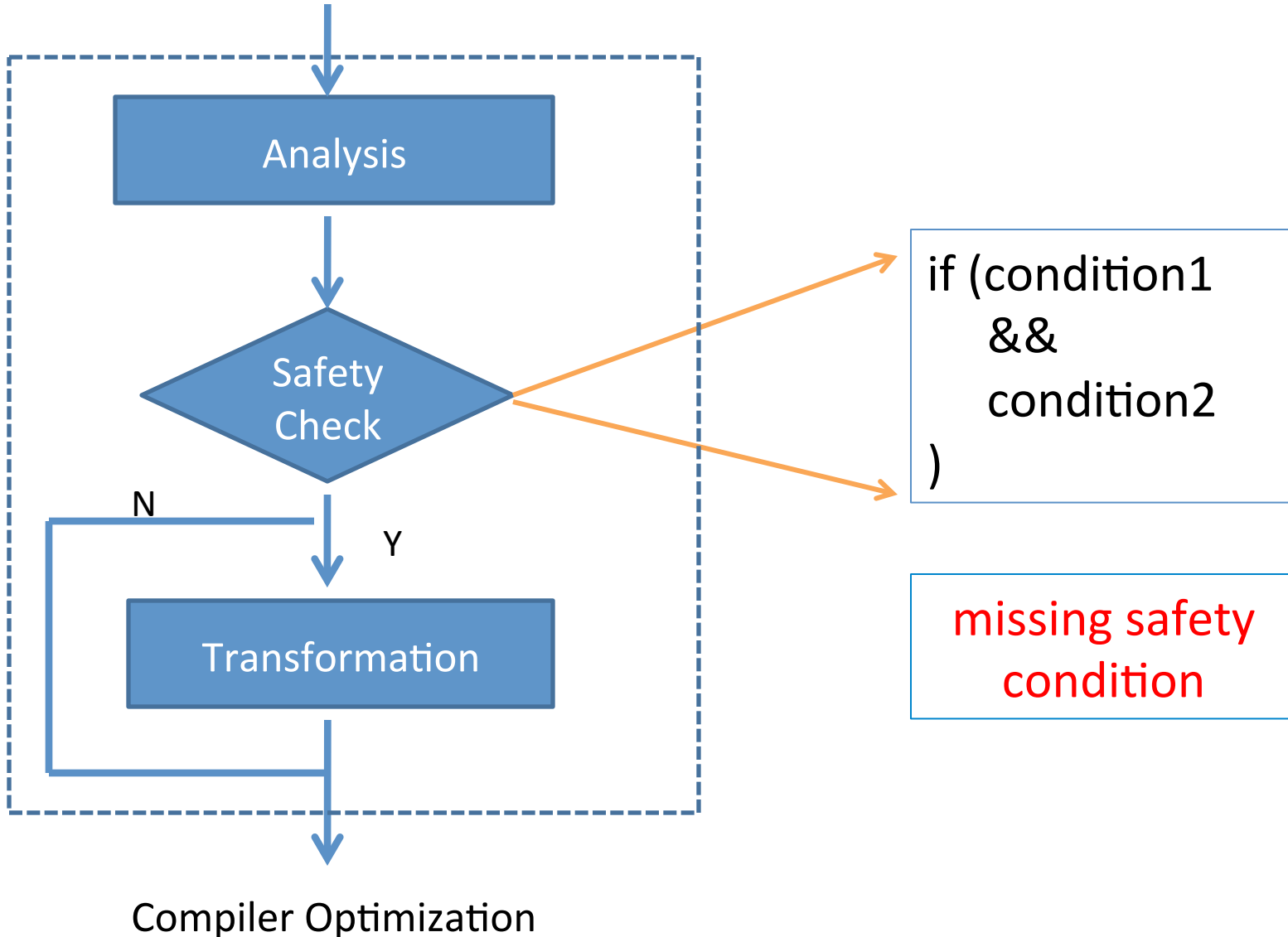
Coverage of GCC



Coverage of LLVM/Clang



Common Compiler Bug Pattern



Optimization Bug

```
void foo (void) {  
    int x;  
    for (x = 0; x < 5; x++) {  
        if (x) continue;  
        if (x) break;  
    }  
    printf ("%d", x);  
}
```

- Bug in LLVM in scalar evolution analysis computed x is 1 after the loop executed

UNDEFINED BEHAVIOR

Example

```
int foo(int a)
{
    return (a+1) > a;
}
```

```
foo: movl $1, %eax
     ret
```

Undefined Behavior

- Executing an erroneous operation
- The program may :
 - fail to compile
 - execute incorrectly
 - crash
 - do exactly what the programmer intended

Undefined Behavior - challenges

- Programmers are not aware of all undefined behavior
- Code may be compiled for a different environment with a different compiler
 - Which undefined behavior are different?

PROJECT IDEAS

1. Add features that are not supported by Csmith
 - C++ constructs
 - Heap allocation
 - Recursive
 - String Operation
 - Use of common libraries
2. Generate programs that takes input
 - Use another fuzzer (constraint-based) to generate inputs to the generated program
3. Generate programs with undefined behavior
 - Automatically understand them
 - Use reduce testcase tools
4. Enhance Csmith by incorporating other fuzzing techniques (mutation, genetic)
5. Apply approach for different languages
6.Your idea...

RESOURCES

- Fuzzer survey

<https://fuzzinginfo.files.wordpress.com/2012/05/dsto-tn-1043-pr.pdf>

- Csmith

Website: <https://embed.cs.utah.edu/csmith/>

paper: <http://www.cs.utah.edu/~regehr/papers/pldi11-preprint.pdf>

- Undefined behavior

– <http://blog.regehr.org/archives/213>