Closer to the Edge: Testing Compilers More Thoroughly by Being Less Conservative About Undefined Behaviour

Karine Even-Mendoza, Cristian Cadar, Alastair F. Donaldson

Imperial College London

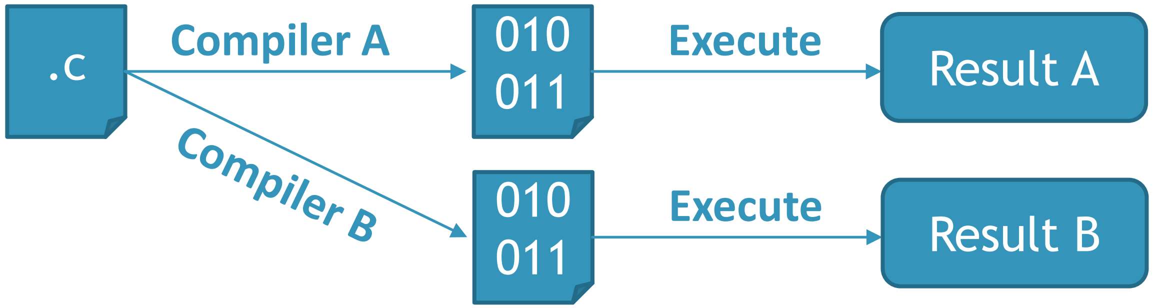
Speaker: Karine Even-Mendoza

ASE-NIER track

22 September, 2020

Fuzzing, Compilers and Undefined Behaviours

- **Compiler testing**: to expose mis-compilation, which can have a broad impact
  - Compiler fuzzers: efficient and successful method for bug hunting
  - Testcase = a program



  - Result A != Result B ➔ mis-compilation

Fuzzing, Compilers and Undefined Behaviours

- **Main challenge**: generating UB-free-programs (to consider a valid bug)

- **Undefined Behaviours** (UB)
  - E.g., UB in C: (i) int x=5÷0; or (ii) int x=INT_MAX + 7;
  - Non-UB-free-program: unpredictable program's result + describe a problem that is not a bug

Fuzzing, Compilers and Undefined Behaviours

- **Fuzzing with Csmith:** successful at finding many bugs in mature compilers [PLDI’11]
  - During/post-generation solutions for UB-free-programs
  - Arithmetic operators: avoid UBs via “safe math” wrappers

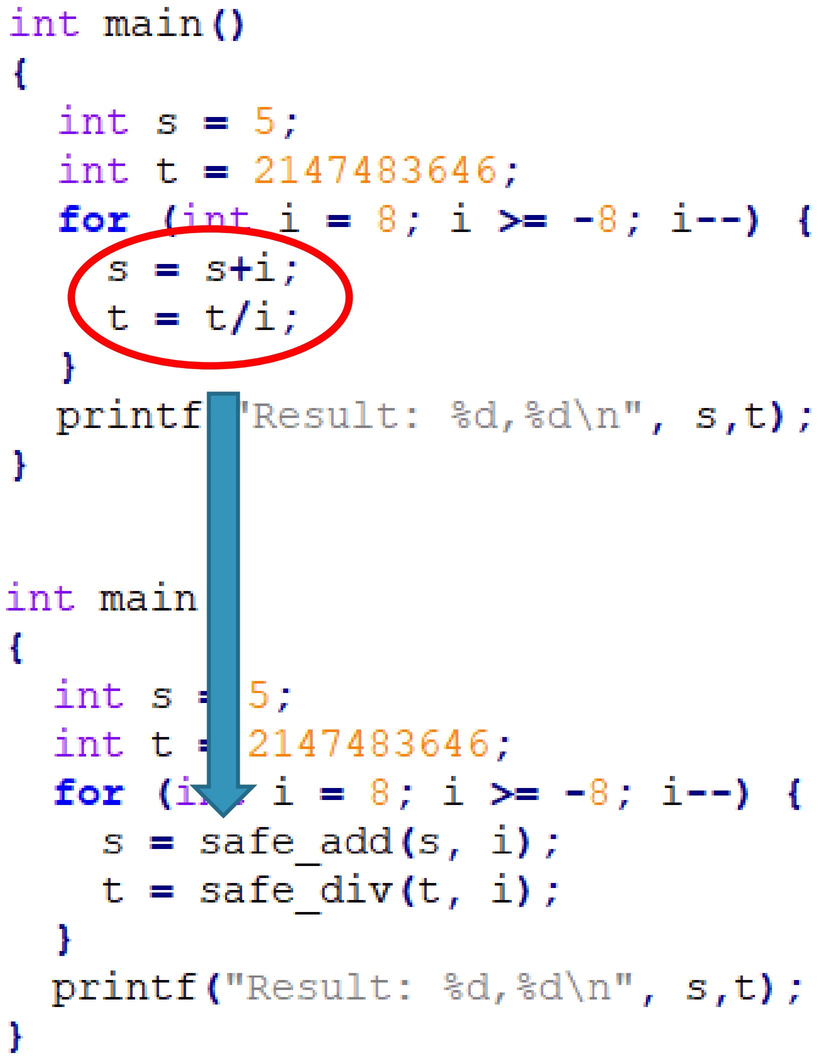
$$unsafe(a, b, \circ)\ ?\ a\ :\ a \circ b$$

Compiler Fuzzers

- **Fuzzing with Csmith:** successful at finding many bugs in mature compilers [PLDI'11]
  - During/post-generation solutions for UB-free-programs
  - Arithmetic operators: avoid UBs via "safe math" wrappers

$$unsafe(a, b, \circ)\ ?\ a\ :\ a \circ b$$

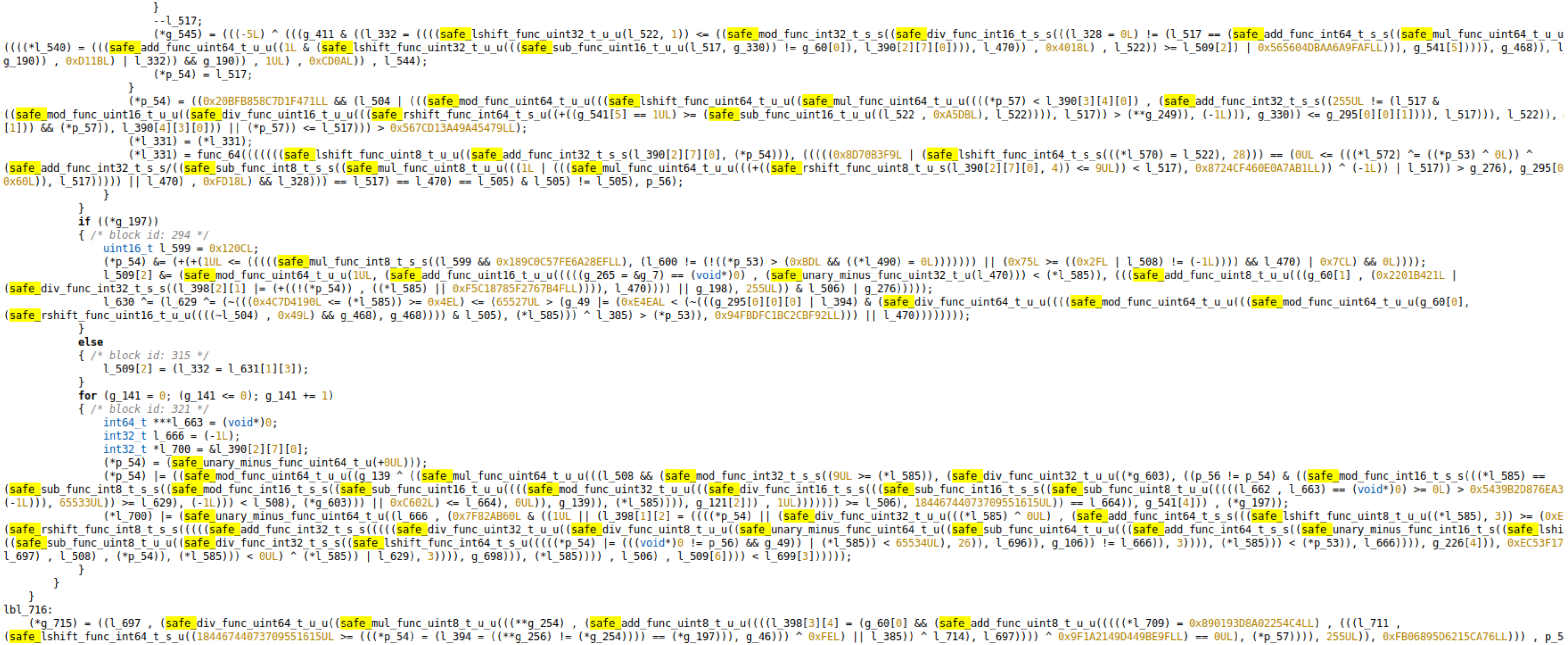
- Compilers became immune to these fuzzers



```
int main()
{
  int s = 5;
  int t = 2147483646;
  for (int i = 8; i >= -8; i--) {
    s = s+i;
    t = t/i;
  }
  printf("Result: %d,%d\n", s,t);
}
```

```
int main
{
  int s = 5;
  int t = 2147483646;
  for (i  i = 8; i >= -8; i--) {
    s = safe_add(s, i);
    t = safe_div(t, i);
  }
  printf("Result: %d,%d\n", s,t);
}
```

Code Fragment of a Csmith Testcase



```
                        }
                        --l_517;
                        (*g_545) = (((-5L) ^ (((g_411 & ((l_332 = ((((safe_lshift_func_uint32_t_u_u(l_522, 1)) <= ((safe_mod_func_int32_t_s_s((safe_div_func_int16_t_s_s(((l_328 = 0L) != (l_517 == (safe_add_func_int64_t_s_s((safe_mul_func_uint64_t_u_u
((((*l_540) = (((safe_add_func_uint64_t_u_u((1L & (safe_lshift_func_uint32_t_u_u(((safe_sub_func_uint16_t_u_u(l_517, g_330)) != g_60[0]), l_390[2][7][0]))), l_470)) , 0x4018L) , l_522)) >= l_509[2]) | 0x565604DBAA6A9FAFLL))), g_541[5])))), g_468)), l
g_190)) , 0xD11BL) | l_332)) && g_190)) , 1UL) , 0xCD0AL)) , l_544);
                        (*p_54) = l_517;
                    }
                    (*p_54) = ((0x20BFB858C7D1F471LL && (l_504 | (((safe_mod_func_uint64_t_u_u(((safe_lshift_func_uint64_t_u_u((safe_mul_func_uint64_t_u_u((((*p_57) < l_390[3][4][0]) , (safe_add_func_int32_t_s_s((255UL != (l_517 &
((safe_mod_func_uint16_t_u_u((safe_div_func_uint16_t_u_u(((safe_rshift_func_int64_t_s_u((+((g_541[5] == 1UL) >= (safe_sub_func_uint16_t_u_u((l_522 , 0xA5DBL), l_522)))), l_517)) > (**g_249)), (-1L))), g_330)) <= g_295[0][0][1]))), l_517))), l_522)),
[1])) && (*p_57)), l_390[4][3][0])) || (*p_57)) <= l_517))) > 0x567CD13A49A45479LL);
                    (*l_331) = (*l_331);
                    (*l_331) = func_64((((((safe_lshift_func_uint8_t_u_u((safe_add_func_int32_t_s_s(l_390[2][7][0], (*p_54))), (((((0x8D70B3F9L | (safe_lshift_func_int64_t_s_s(((*l_570) = l_522), 28))) == (0UL <= (((*l_572) ^= ((*p_53) ^ 0L)) ^
(safe_add_func_int32_t_s_s/((safe_sub_func_int8_t_s_s((safe_mul_func_uint8_t_u_u(((1L | (((safe_mul_func_uint64_t_u_u(((+((safe_rshift_func_uint8_t_u_s(l_390[2][7][0], 4)) <= 9UL)) < l_517), 0x8724CF460E0A7AB1LL)) ^ (-1L)) | l_517)) > g_276), g_295[0
0x60L)), l_517))))) || l_470) , 0xFD18L) && l_328))) == l_517) == l_470) == l_505) & l_505) != l_505), p_56);
                }
            }
            if ((*g_197))
            { /* block id: 294 */
                uint16_t l_599 = 0x120CL;
                (*p_54) &= (+(+(1UL <= (((((safe_mul_func_int8_t_s_s((l_599 && 0x189C0C57FE6A28EFLL), (l_600 != (!((*p_53) > (0xBDL && ((*l_490) = 0L))))))) || (0x75L >= ((0x2FL | l_508) != (-1L)))) && l_470) | 0x7CL) && 0L))));
                l_509[2] &= (safe_mod_func_uint64_t_u_u(1UL, (safe_add_func_uint16_t_u_u(((((g_265 = &g_7) == (void*)0) , (safe_unary_minus_func_uint32_t_u(l_470))) < (*l_585)), (((safe_add_func_uint8_t_u_u(((g_60[1] , (0x2201B421L |
(safe_div_func_int32_t_s_s((l_398[2][1] |= (+((!(*p_54)) , ((*l_585) || 0xF5C18785F2767B4FLL)))), l_470)))) || g_198), 255UL)) & l_506) | g_276)))));
                l_630 ^= (l_629 ^= (~(((0x4C7D4190L <= (*l_585)) >= 0x4EL) <= (65527UL > (g_49 |= (0xE4EAL < (~(((g_295[0][0][0] | l_394) & (safe_div_func_uint64_t_u_u((((safe_mod_func_uint64_t_u_u(((safe_mod_func_uint64_t_u_u(g_60[0],
(safe_rshift_func_uint16_t_u_u((((~l_504) , 0x49L) && g_468), g_468)))) & l_505), (*l_585))) ^ l_385) > (*p_53)), 0x94FBDFC1BC2CBF92LL))) || l_470))))))));
            }
            else
            { /* block id: 315 */
                l_509[2] = (l_332 = l_631[1][3]);
            }
            for (g_141 = 0; (g_141 <= 0); g_141 += 1)
            { /* block id: 321 */
                int64_t ***l_663 = (void*)0;
                int32_t l_666 = (-1L);
                int32_t *l_700 = &l_390[2][7][0];
                (*p_54) = (safe_unary_minus_func_uint64_t_u(+0UL));
                (*p_54) |= ((safe_mod_func_uint64_t_u_u((g_139 ^ ((safe_mul_func_uint64_t_u_u(((l_508 && (safe_mod_func_int32_t_s_s((9UL >= (*l_585)), (safe_div_func_uint32_t_u_u((*g_603), ((p_56 != p_54) & ((safe_mod_func_int16_t_s_s(((*l_585) ==
(safe_sub_func_int8_t_s_s((safe_mod_func_int16_t_s_s((safe_sub_func_uint16_t_u_u((((safe_mod_func_uint32_t_u_u(((safe_div_func_int16_t_s_s(((safe_sub_func_int16_t_s_s((safe_sub_func_uint8_t_u_u(((((l_662 , l_663) == (void*)0) >= 0L) > 0x5439B2D876EA3
(-1L))), 65533UL)) >= l_629), (-1L))) < l_508), (*g_603))) || 0xC602L) <= l_664), 0UL)), g_139)), (*l_585)))), g_121[2])) , 1UL))))))) >= l_506), 18446744073709551615UL)) == l_664)), g_541[4])) , (*g_197));
                (*l_700) |= (safe_unary_minus_func_uint64_t_u((l_666 , (0x7F82AB60L & ((1UL || (l_398[1][2] = ((((*p_54) || (safe_div_func_uint32_t_u_u(((*l_585) ^ 0UL) , (safe_add_func_int64_t_s_s(((safe_lshift_func_uint8_t_u_u((*l_585), 3)) >= (0xE
(safe_rshift_func_int8_t_s_s(((((safe_add_func_int32_t_s_s(((((safe_div_func_uint32_t_u_u((safe_div_func_uint8_t_u_u((safe_unary_minus_func_uint64_t_u((safe_sub_func_uint64_t_u_u(((safe_add_func_int64_t_s_s((safe_unary_minus_func_int16_t_s((safe_lshi
((safe_sub_func_uint8_t_u_u((safe_div_func_int32_t_s_s((safe_lshift_func_int64_t_s_u(((((*p_54) |= (((void*)0 != p_56) && g_49)) | (*l_585)) < 65534UL), 26)), l_696)), g_106)) != l_666)), 3)))), (*l_585))) < (*p_53)), l_666)))), g_226[4])), 0xEC53F17
l_697) , l_508) , (*p_54)), (*l_585))) < 0UL) ^ (*l_585)) | l_629), 3)))), g_698))), (*l_585)))) , l_506) , l_509[6]))) < l_699[3]))))));
            }
        }
    }
lbl_716:
    (*g_715) = ((l_697 , (safe_div_func_uint64_t_u_u((safe_mul_func_uint8_t_u_u(((**g_254) , (safe_add_func_uint8_t_u_u((((l_398[3][4] = (g_60[0] && (safe_add_func_uint8_t_u_u(((((*l_709) = 0x890193D8A02254C4LL) , (((l_711 ,
(safe_lshift_func_int64_t_s_u((18446744073709551615UL >= (((*p_54) = (l_394 = ((**g_256) != (*g_254)))) == (*g_197))), g_46))) ^ 0xFEL) || l_385)) ^ l_714), l_697)))) ^ 0x9F1A2149D449BE9FLL) == 0UL), (*p_57)))), 255UL)), 0xFB06895D6215CA76LL))) , p_5
```

What Code Can We Generate?

- **Observation**

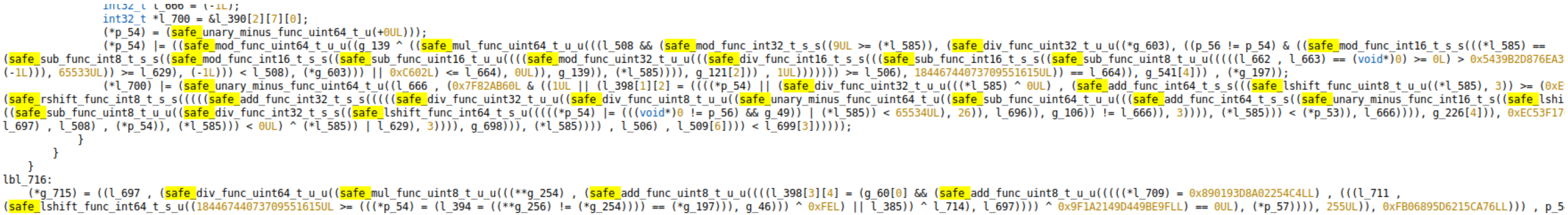
  None of these operators found outside ternary operator + use several times in a statement

- **Problem**

  E.g., some of the code optimizations in the compiler can be inapplicable if enclosed in these checks

- **Hypothesis**

  Safe math wrappers everywhere ➔ limits the form of programs we can generate



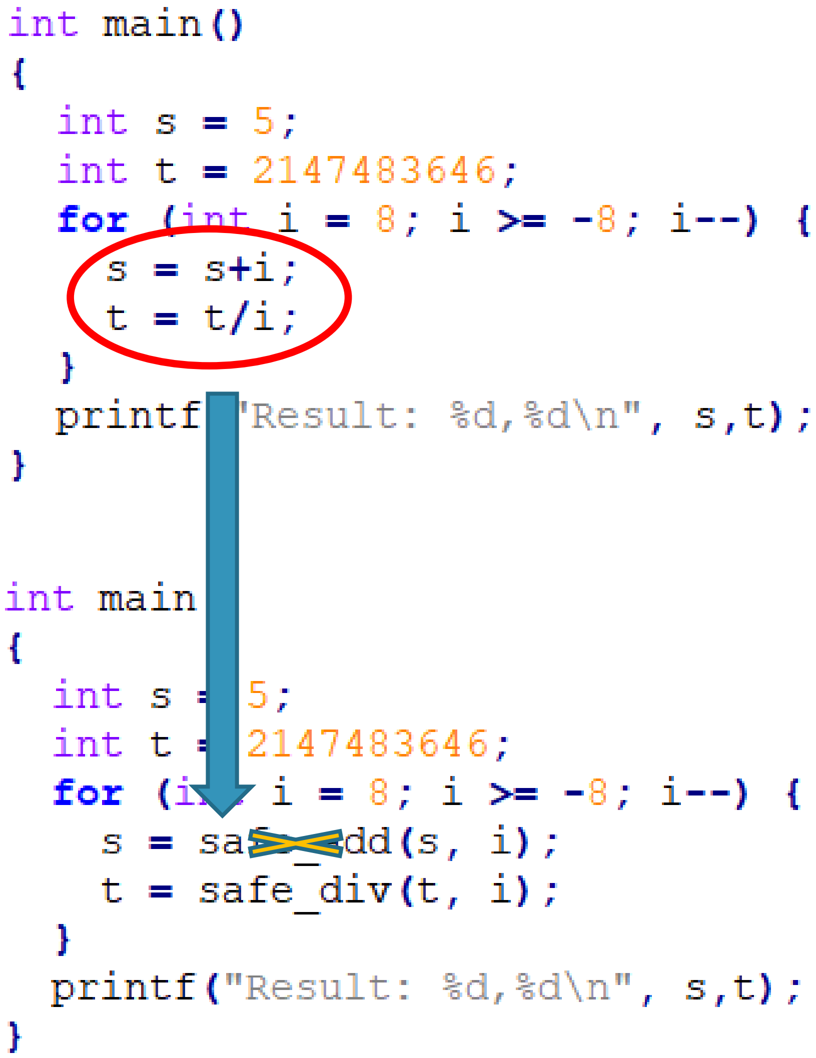
```
                int32_t l_666 = (-1L);
                int32_t *l_700 = &l_390[2][7][0];
                (*p_54) = (safe_unary_minus_func_uint64_t_u(+0UL));
                (*p_54) |= ((safe_mod_func_uint64_t_u_u((g_139 ^ ((safe_mul_func_uint64_t_u_u(((l_508 && (safe_mod_func_int32_t_s_s((9UL >= (*l_585)), (safe_div_func_uint32_t_u_u((*g_603), ((p_56 != p_54) & ((safe_mod_func_int16_t_s_s(((*l_585) ==
(safe_sub_func_int8_t_s_s((safe_mod_func_int16_t_s_s((safe_sub_func_uint16_t_u_u((((safe_mod_func_uint32_t_u_u(((safe_div_func_int16_t_s_s(((safe_sub_func_int16_t_s_s((safe_sub_func_uint8_t_u_u(((((l_662 , l_663) == (void*)0) >= 0L) > 0x5439B2D876EA3
(-1L))), 65533UL)) >= l_629), (-1L))) < l_508), (*g_603))) || 0xC602L) <= l_664), 0UL)), g_139)), (*l_585)))), g_121[2])) , 1UL))))))) >= l_506), 18446744073709551615UL)) == l_664)), g_541[4])) , (*g_197));
                (*l_700) |= (safe_unary_minus_func_uint64_t_u((l_666 , (0x7F82AB60L & ((1UL || (l_398[1][2] = ((((*p_54) || (safe_div_func_uint32_t_u_u(((*l_585) ^ 0UL) , (safe_add_func_int64_t_s_s(((safe_lshift_func_uint8_t_u_u((*l_585), 3)) >= (0xE
(safe_rshift_func_int8_t_s_s(((((safe_add_func_int32_t_s_s(((((safe_div_func_uint32_t_u_u((safe_div_func_uint8_t_u_u((safe_unary_minus_func_uint64_t_u((safe_sub_func_uint64_t_u_u(((safe_add_func_int64_t_s_s((safe_unary_minus_func_int16_t_s((safe_lshi
((safe_sub_func_uint8_t_u_u((safe_div_func_int32_t_s_s((safe_lshift_func_int64_t_s_u(((((*p_54) |= (((void*)0 != p_56) && g_49)) | (*l_585)) < 65534UL), 26)), l_696)), g_106)) != l_666)), 3)))), (*l_585))) < (*p_53)), l_666)))), g_226[4])), 0xEC53F17
l_697) , l_508) , (*p_54)), (*l_585))) < 0UL) ^ (*l_585)) | l_629), 3)))), g_698))), (*l_585)))) , l_506) , l_509[6]))) < l_699[3]))))));
            }
        }
    }
lbl_716:
    (*g_715) = ((l_697 , (safe_div_func_uint64_t_u_u((safe_mul_func_uint8_t_u_u(((**g_254) , (safe_add_func_uint8_t_u_u((((l_398[3][4] = (g_60[0] && (safe_add_func_uint8_t_u_u(((((*l_709) = 0x890193D8A02254C4LL) , (((l_711 ,
(safe_lshift_func_int64_t_s_u((18446744073709551615UL >= (((*p_54) = (l_394 = ((**g_256) != (*g_254)))) == (*g_197))), g_46))) ^ 0xFEL) || l_385)) ^ l_714), l_697)))) ^ 0x9F1A2149D449BE9FLL) == 0UL), (*p_57)))), 255UL)), 0xFB06895D6215CA76LL))) , p_5
```

Compiler Fuzzers

- **Fuzzing with Csmith:** successful at finding many bugs in mature compilers [PLDI'11]
  - During/post-generation solutions for UB-free-programs
  - Arithmetic operators: avoid UBs via "safe math" wrappers

$$if(b == 0)\ ?\ a\ :\ a/b$$

- Compilers became immune to these fuzzers
- **Observation + Hypothesis** ➔ found new bugs in GCC

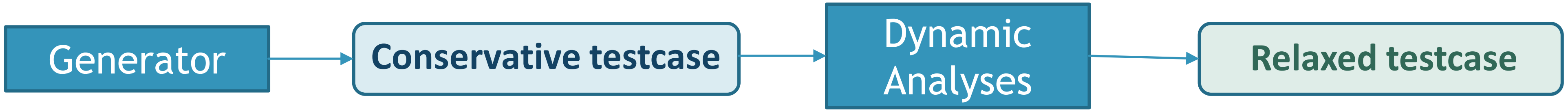


```
int main()
{
  int s = 5;
  int t = 2147483646;
  for (int i = 8; i >= -8; i--) {
    s = s+i;
    t = t/i;
  }
  printf("Result: %d,%d\n", s,t);
}
```

```
int main()
{
  int s = 5;
  int t = 2147483646;
  for (int i = 8; i >= -8; i--) {
    s = safe_add(s, i);
    t = safe_div(t, i);
  }
  printf("Result: %d,%d\n", s,t);
}
```

Being Less Conservative

- **CsmithEdge**
  - Modifies Csmith’s programs to create more interesting testcases
  - Post-gen. dynamic analysis: to identify and eliminate redundant Csmith’s *safe math* wrappers

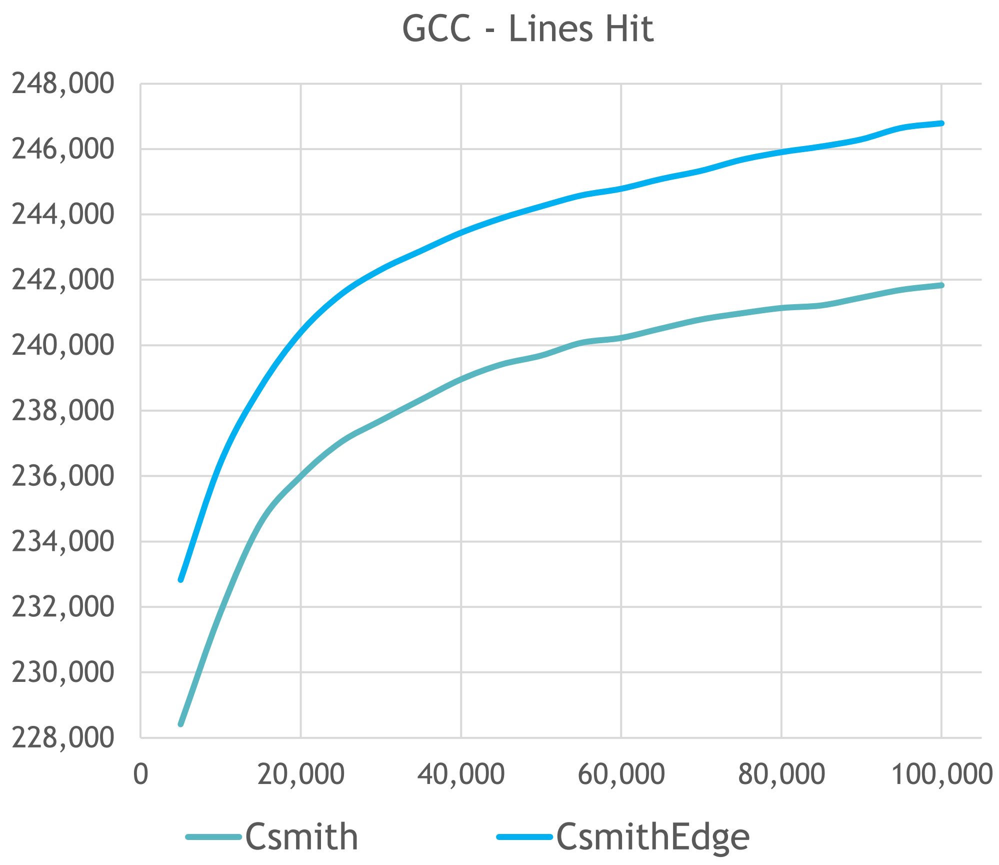


```
int main()
{
  int s = 5;
  int t = 2147483646;
  for (int i = 8; i >= -8; i--) {
    s = safe_add(s, i);
    t = safe_div(t, i);
  }
  printf("Result: %d,%d\n", s,t);
}
```

```
int main()
{
  int s = 5;
  int t = 2147483646;
  for (int i = 8; i >= -8; i--) {
    s = s+i;
    t = safe_div(t, i);
  }
  printf("Result: %d,%d\n", s,t);
}
```

Preliminary Evaluation

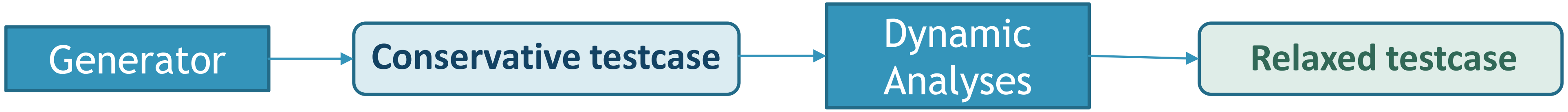
- **Two new bugs**: GCC-10, P2 normal, tree-optimisation, fixed promptly, discovered only by CsmithEdge
  - GCC Bug #1: Skipping tree-side-effect evaluation of operator’s 2nd argument
  - GCC Bug #2: Skipping tree-side-effects on internal calls
  - We reported additional bugs since then
- **Line coverage**: 100,000 test-cases, compared against Csmith, with 4k lines uniquely-covered



Line coverage in GCC-10.0.1

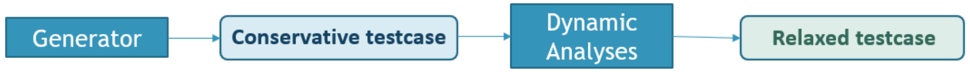
Future Work

- **Bug-hunting**: trying different compilers (e.g., LLVM or Microsoft Visual Studio)
- **Post-generation/during testing**: extends the possible modification we allow in post-generation
- **During generation:** relax restrictions + (after) detect and discard those with UBs
  - E.g., can skip variables initialization when declared, or allow null pointers

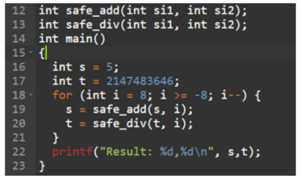


Being Less Conservative

- **CsmithEdge**
  - Modifies Csmith's programs to create more interesting testcases
  - Post-gen. dynamic analysis: to identify and eliminate redundant Csmith's *safe math* wrappers

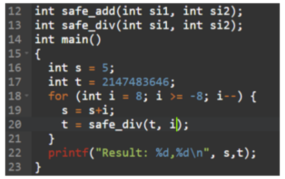


1



```
int safe_add(int si1, int si2);
int safe_div(int si1, int si2);
int main()
{
  int s = 5;
  int t = 2147483646;
  for (int i = 8; i >= -8; i--) {
    s = safe_add(s, i);
    t = safe_div(t, i);
  }
  printf("Result: %d,%d\n", s,t);
}
```

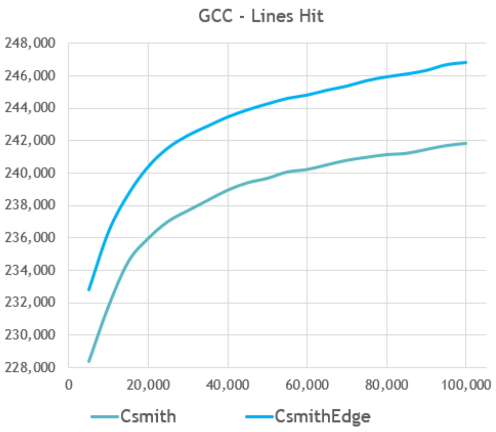
2



```
int safe_add(int si1, int si2);
int safe_div(int si1, int si2);
int main()
{
  int s = 5;
  int t = 2147483646;
  for (int i = 8; i >= -8; i--) {
    s = s+i;
    t = safe_div(t, i);
  }
  printf("Result: %d,%d\n", s,t);
}
```

Preliminary Evaluation

- **Two new bugs**: GCC-10, P2 normal, tree-optimisation, fixed promptly, discovered only by CsmithEdge
  - GCC Bug #1: Skipping tree-side-effect evaluation of operator's 2nd argument
  - GCC Bug #2: Skipping tree-side-effects on internal calls
  - We reported additional bugs since then
- **Line coverage**: 100,000 test-cases, compared against Csmith, with <u>4k lines</u> uniquely-covered



Line coverage in GCC-10.0.1