CS348
Safe Programming Systems

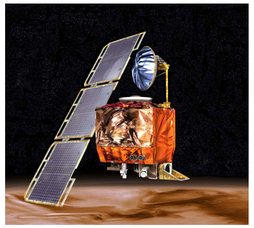
Min Suk Kang **& Kihong Heo**

KAIST

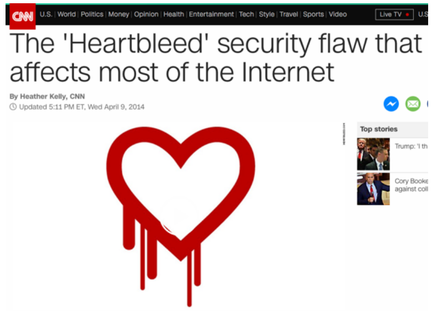
Impact of Poor Software Quality

The Patriot Missile (1991)
Floating-point roundoff
28 soldiers died

The Ariane-5 Rocket (1996)
Integer Overflow
$100M



NASA's Mars Climate Orbiter (1999)
Meters-Inches Miscalculation
$125M



Cost of Software Quality Assurance

*"We have as **many testers** as we have developers.*
*And testers spend **all their time testing**, and developers spend*
***half their time testing**. We're more of a testing, a quality software*
organization than we're a software organization"
- Bill Gates

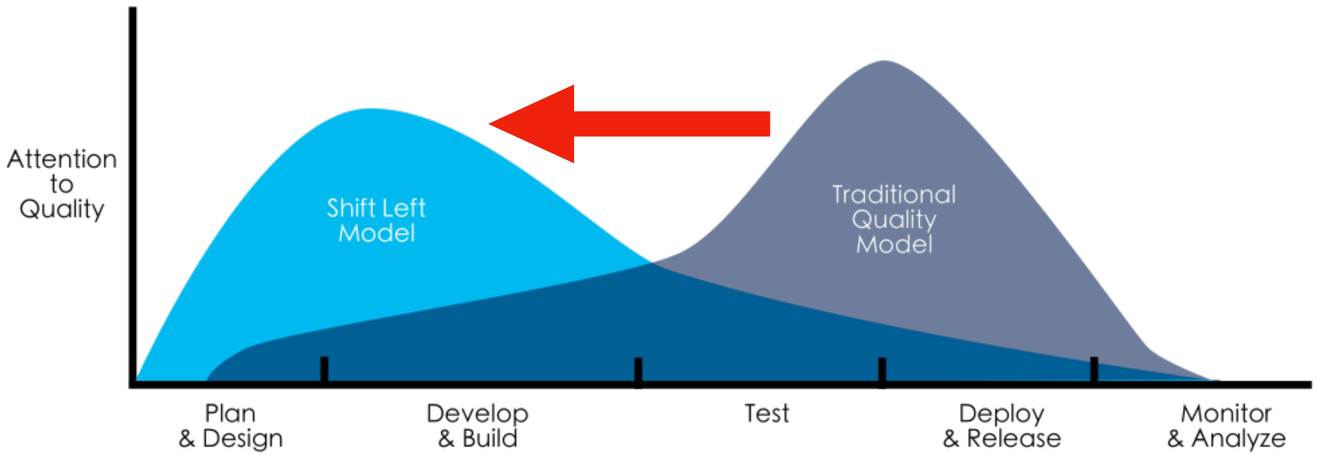
Discovering Software Errors

- Very important as SW is eating the world!
- Key issue: how to detect SW errors as early as possible?

COST OF A SOFTWARE BUG

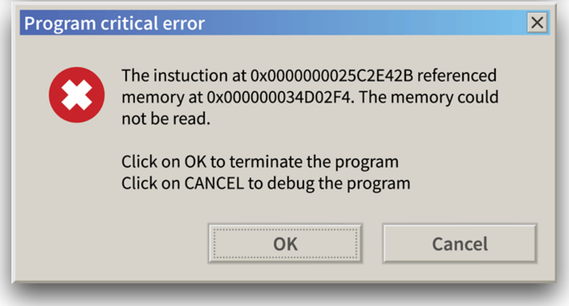
$100	$1,500	$10,000
If found in Gathering Requirements phase	If found in QA testing phase	If found in Production

- IBM Systems Sciences Institute, 2015



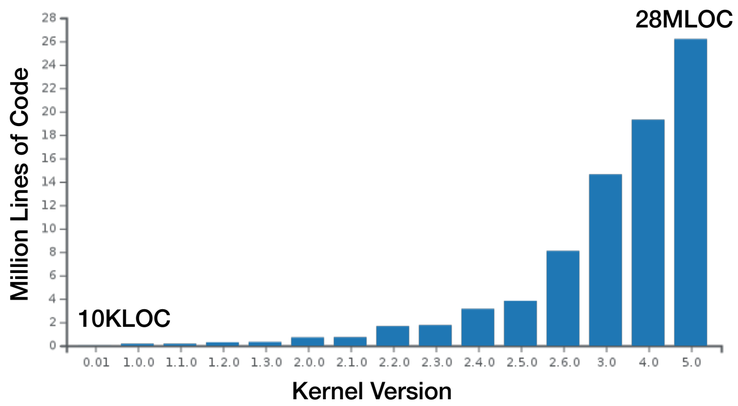
SW Error (a.k.a SW Bugs)

- What?
  - SW runs and produces outputs unexpectedly
- Why?
  - Incorrectly written code by human or AI

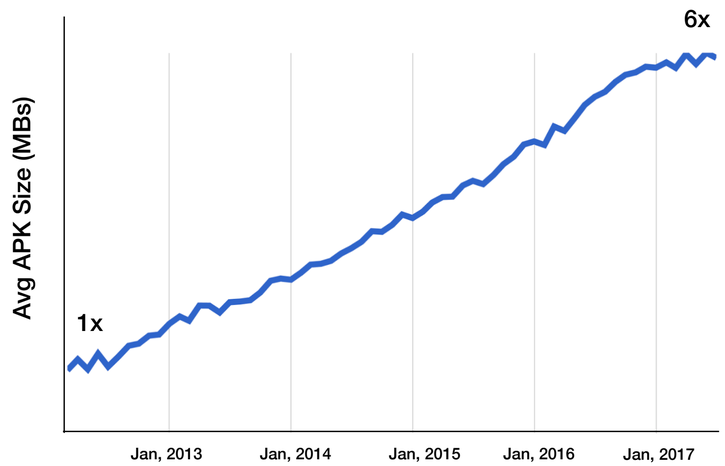


Why Software Fails?

Size of Linux Kernel



Avg. Size of Android Apps



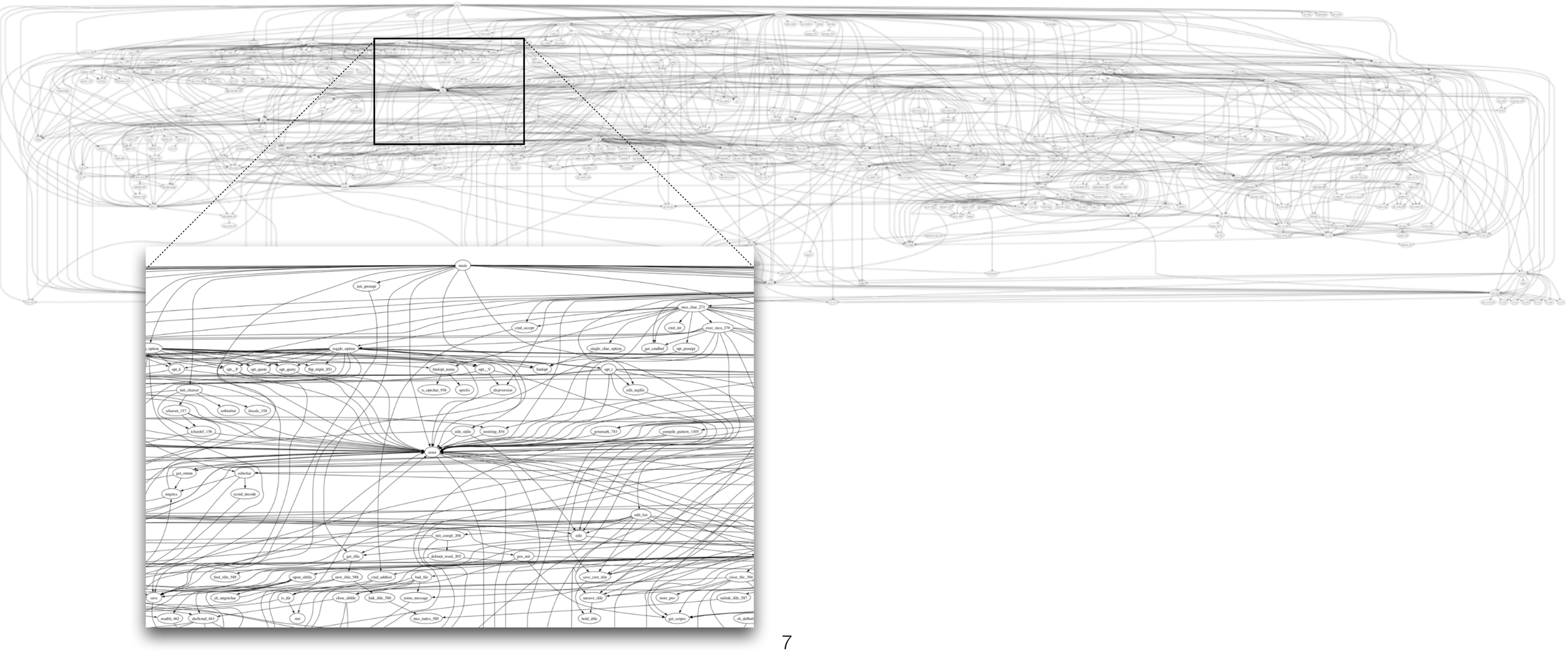
X

GitHub

10M+ New Developers
44M+ New Repositories
87M+ New Pull Requests
in 2019

Software Complexity

less-382 (23,822 LOC)



Automated Software Verification

Q: How to **formally estimate** software behavior automatically before its execution?

SW Error Detection Level 1

- Lexical analyzer & parser (1970s)
- SW errors = ill-formed programs
- Already in the mainstream

```
function getAPlusInCS348()
  1. praticipate atcivly
  2. ask frqeuently
  3. tihnk carfeully
```

SW Error Detection Level 2

- Type systems (1990s)
- SW error = well-formed but ill-typed program
- Gradually adopted in the mainstream (e.g., OCaml, Haskell, Rust, F#, Scala, etc)

```
function getAPlusInCs348()
  1. participate actively in the classic
  2. ask frequently to the proof
  3. think carefully by yours
```

Success Stories

x

*"Over the course of its lifetime, there have been **69 security bugs** in **Firefox**'s style component. If we'd had a time machine and could have **written this component in Rust** from the start, **51 (73.9%) of these bugs** would not have been possible."*

- Diane Hosfelt (SW Engineer at Mozilla), 2019

x

*"As our system has grown, a lot of the logic in our Ruby system sort of replicates a **type system**. … It is a shame to have to write all that when there is a solution that has existed in the world of programming languages for decades now."*

- Alex Payne (API Lead at Twitter), 2009

x ERLANG

*"**WhatsApp** uses a surprisingly **small amount of engineers** for the billions of users it caters to on a daily basis. How do they manage this? **Erlang** was built to solve very specific problems, in particular scaling a large system with it still remaining **highly reliable**."*

- Erlang Solutions, 2018

x OCaml

*"There is, however, a **surprisingly wide swath of bugs** against which the **type system** is effective, including many bugs that are quite hard to get at through testing."*

- Yaron Minsky (SW Engineer at Janestreet), 2018

SW Error Detection Level 3

- SW error = well-formed and well-typed but ill-behaved program
- Actively developed in academia and industry
- Unfortunately, no general and perfect solution! Why?

```
function getAPlusInCS348()
  1. participate actively in the class
  2. ask frequently to the professor
  3. think carefully by yourself
```

class == CS101?

10^{10} qstns/hr?

morality < 0 &&
become a hacker?

A Hard Limit: Undecidability

Theorem (Rice's theorem). Any non-trivial semantic properties are undecidable.

- Non-trivial property: worth the effort of designing a program analyzer for
  - E.g., buffer overrun, integer overflow, etc
- Undecidable? If decidable, it can solves the Halting problem!

HP: Given a Turing machine T and an input i, does T eventually halt on i?

Undecidable: There is no Turing machine that can solve HP!

Informal Proof of Undecidability of HP

HP: Given a Turing machine T and an input i, does T eventually halt on i?

- Assume $H(T, i)$ returns true or false
- Let $F(x) = \text{if } H(x, x) \text{ then } \texttt{loop}() \text{ else } \texttt{halt}()$
- Does $F(F)$ terminate?

Informal Proof of Rice's Theorem

- Fact: HP is undecidable
- An analyzer **A** for a property: "*This program always prints 1 and finishes*"
- Given a program **P**, generate **P' =** "**P**; print 1;"
- Analyze **P'** using **A: A(P')**
  - **A(P')** says "Yes": **P** halts,
  - **A(P')** says "No": **P** does not halt
- HP is decidable if we use **A** : contradiction!

Toward Computability

Undecidable

⇒ Automatic, terminating, and exact reasoning is impossible

⇒ If we give up one of them, it is computable!

- **Manual** rather than **automatic**: assisted proving
  - require expertise and manual effort
- **Possibly nonterminating** rather than **terminating**: model checking, testing
  - require stopping mechanisms such as timeout
- **Approximate** rather than **exact**: static analysis
  - report spurious results

Soundness and Completeness

- Given a semantic property $\mathscr{P}$, and an analysis tool **A**

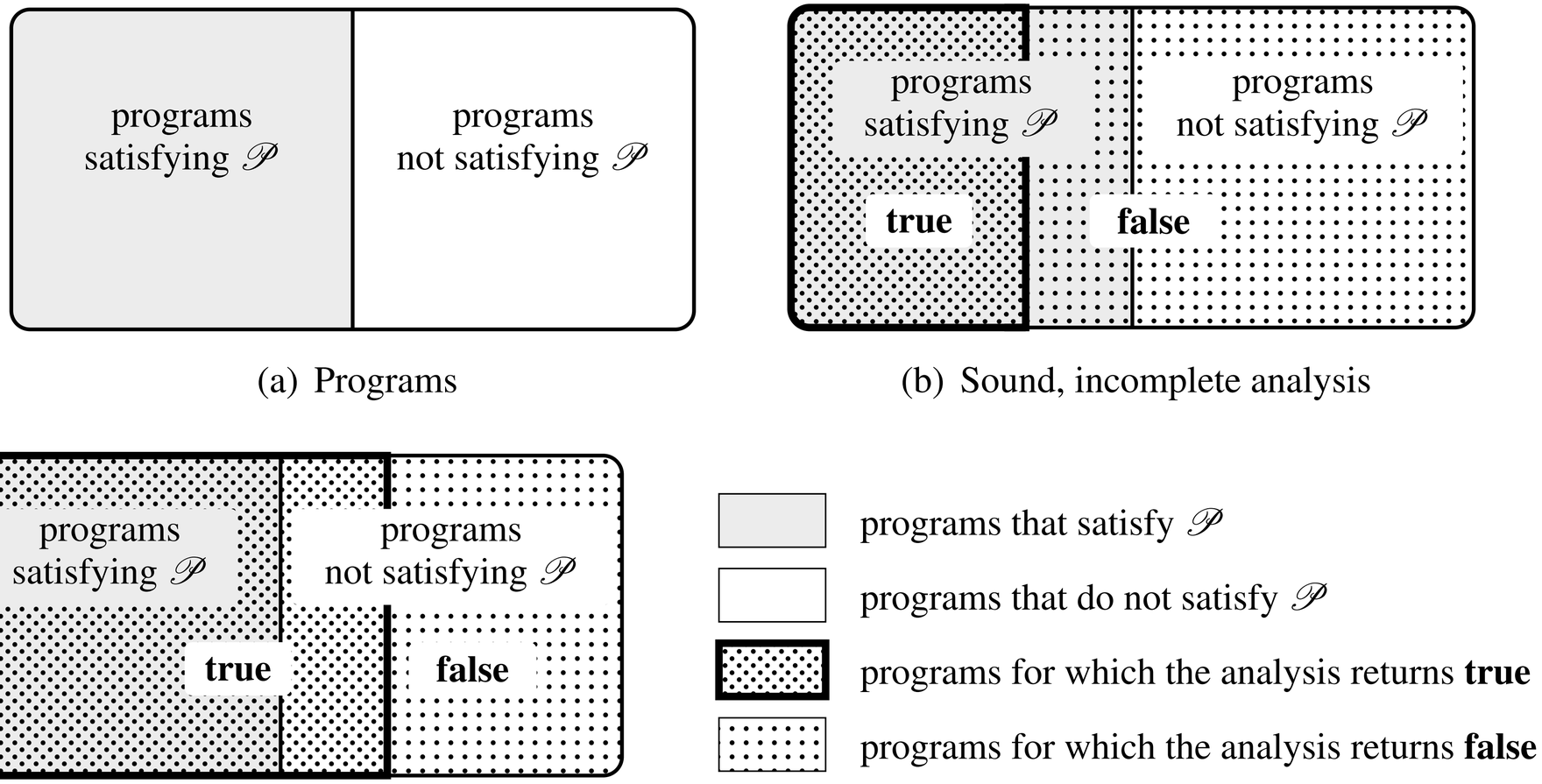
- If **A** were perfectly accurate,

For all program p**, A(**p**) =** $\mathbf{true}$ $\Longleftrightarrow$ p satisfies $\mathscr{P}$

For all program p**, A(**p**) =** $\mathbf{true}$ $\Rightarrow$ p satisfies $\mathscr{P}$ **(soundness)**

For all program p**, A(**p**) =** $\mathbf{true}$ $\Leftarrow$ p satisfies $\mathscr{P}$ **(completeness)**

Soundness and Completeness



(a) Programs

(b) Sound, incomplete analysis

(c) Unsound, complete analysis

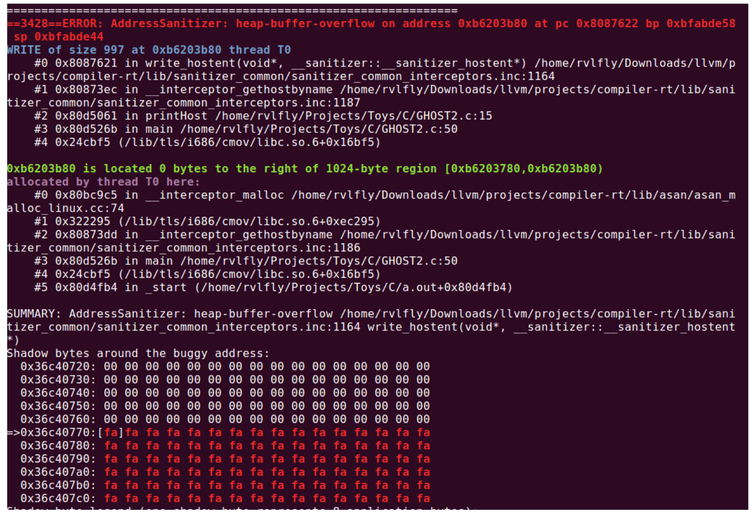
(d) Legend

Testing

- Check a set of **finite executions**
  - e.g., random testing (a.k.a fuzzing)
- In general, **unsound yet complete**
  - Unsound: cannot prove the absence of errors
  - Complete: produce counterexamples (i.e., erroneous inputs)
- Example: Google's oss-fuzz (https://github.com/google/oss-fuzz)

Runtime Checking (Dynamic Analysis)

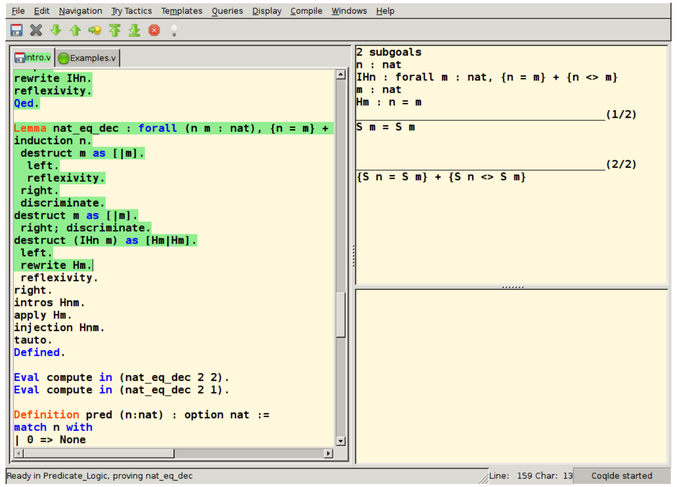
- Compilers instrument code to invoke checking code
  - E.g., add bound-checking code before every buffer access
- E.g., LLVM's sanitizers (ASAN, UBSAN, MEMSAN, etc), Java exceptions
- Involve compiler support, runtime supports, and runtime overhead



```
=================================================================
==3428==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xb6203b80 at pc 0x8087622 bp 0xbfabde58
 sp 0xbfabde44
WRITE of size 997 at 0xb6203b80 thread T0
    #0 0x8087621 in write_hostent(void*, __sanitizer::__sanitizer_hostent*) /home/rvlfly/Downloads/llvm/p
rojects/compiler-rt/lib/sanitizer_common/sanitizer_common_interceptors.inc:1164
    #1 0x80873ec in __interceptor_gethostbyname /home/rvlfly/Downloads/llvm/projects/compiler-rt/lib/sani
tizer_common/sanitizer_common_interceptors.inc:1187
    #2 0x80d5061 in printHost /home/rvlfly/Projects/Toys/C/GHOST2.c:15
    #3 0x80d526b in main /home/rvlfly/Projects/Toys/C/GHOST2.c:50
    #4 0x24cbf5 (/lib/tls/i686/cmov/libc.so.6+0x16bf5)

0xb6203b80 is located 0 bytes to the right of 1024-byte region [0xb6203780,0xb6203b80)
allocated by thread T0 here:
    #0 0x80bc9c5 in __interceptor_malloc /home/rvlfly/Downloads/llvm/projects/compiler-rt/lib/asan/asan_m
alloc_linux.cc:74
    #1 0x322295 (/lib/tls/i686/cmov/libc.so.6+0xec295)
    #2 0x80873dd in __interceptor_gethostbyname /home/rvlfly/Downloads/llvm/projects/compiler-rt/lib/sani
tizer_common/sanitizer_common_interceptors.inc:1186
    #3 0x80d526b in main /home/rvlfly/Projects/Toys/C/GHOST2.c:50
    #4 0x24cbf5 (/lib/tls/i686/cmov/libc.so.6+0x16bf5)
    #5 0x80d4fb4 in _start (/home/rvlfly/Projects/Toys/C/a.out+0x80d4fb4)

SUMMARY: AddressSanitizer: heap-buffer-overflow /home/rvlfly/Downloads/llvm/projects/compiler-rt/lib/sani
tizer_common/sanitizer_common_interceptors.inc:1164 write_hostent(void*, __sanitizer::__sanitizer_hostent
*)
Shadow bytes around the buggy address:
  0x36c40720: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x36c40730: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x36c40740: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x36c40750: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x36c40760: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x36c40770:[fa]fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x36c40780: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x36c40790: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x36c407a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x36c407b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x36c407c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
```

Assisted Proving

- Machine-assisted proof techniques
  - Relying on user-provided proofs or invariants
  - Using proof assistants (e.g., Coq, Isabelle/HOL)
- **Sound and complete** (up to the ability of the proof assistant)
  - require manual effort / expertise
- Example: CompCert (verified C compiler), seL4 (verified microkernel)



Static Analysis

- **Over-approximate** (not exact) the set of all program behavior
- In general, **sound and automatic, but incomplete**
  - May have spurious results
- Based on a foundational theory : Abstract interpretation
- Variants:
  - under-approximating static analysis: automatic, complete, unsound
  - bug finder: automatic, unsound, incomplete, and heuristics
- Example: type systems, ASTREE, Facebook Infer, Sparrow, etc

Emerging Technologies

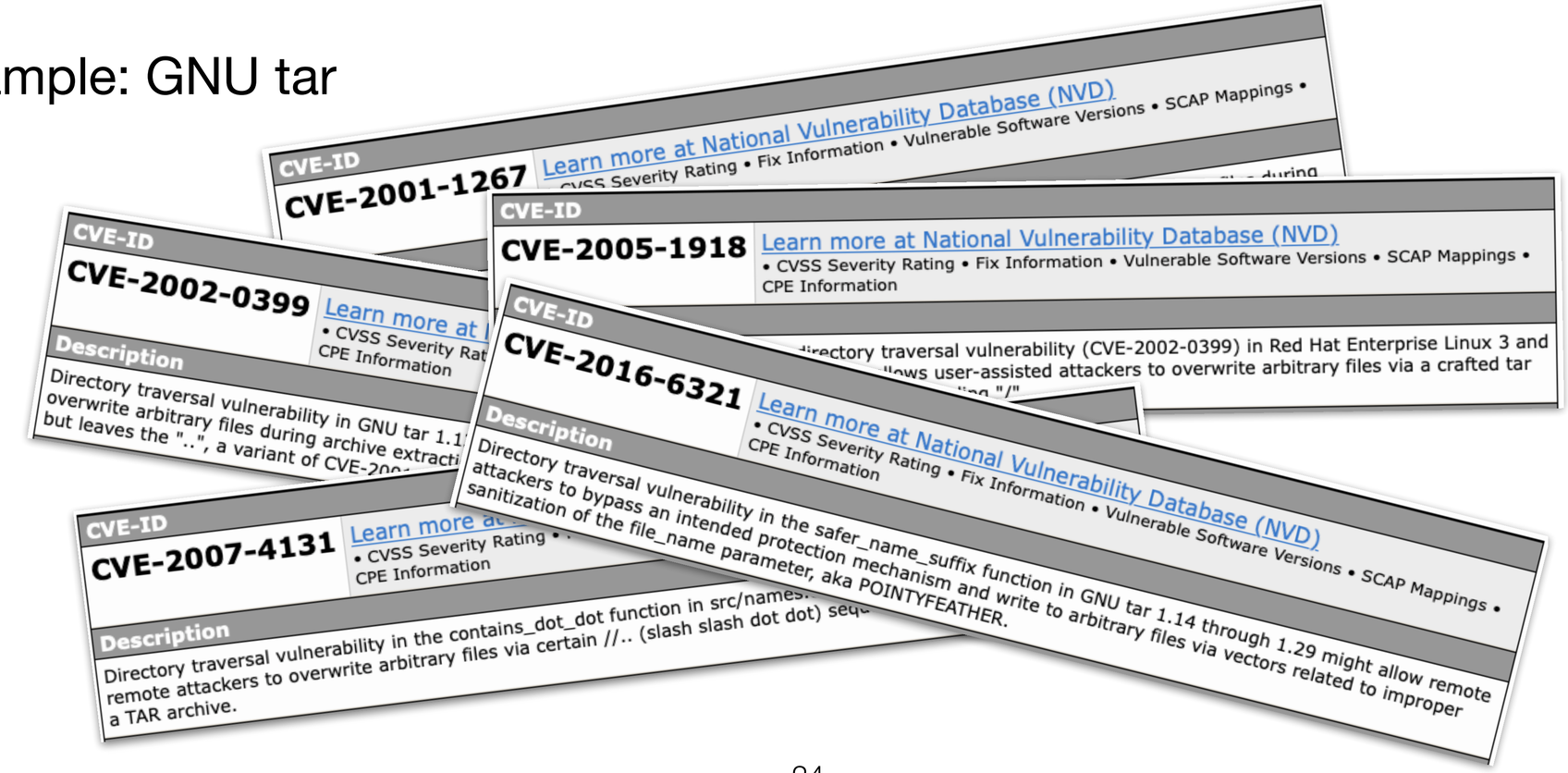
Q: Why do we have to spend so much time finding bugs in the existing code?

Q-1: Why not just remove it if unused?

Q-2: Why not automatically generate in the first place?

Software Bloat

- SW becomes bigger and bigger! Why?
- Consequences of SW bloat: performance, maintainability, security, etc
- An example: GNU tar



Program Debloating

- More code = more vulnerable!
- Aggressively remove unnecessary code so that reduce attack surface
- For example, GNU tar with 97 cmd line options

```
int absolute_names;
int ignore_zeros_option;
struct tar_stat_info stat_info;

char *safer_name_suffix (char *file_name, int link_target) {
    int prefix_len;
    char *p;

    if (absolute_names) {
        p = file_name;
    } else {
       /* CVE-2016-6321 */
       /* Incorrect sanitization if “file_name” contains ".." */
       ...
    }
    ...
    return p;
}

void extract_archive() {
    char *file_name = safer_name_suffix(stat_info.file_name, 0);
    /* Overwrite “file_name” if exists */
    ...
}

void list_archive() { ... }
```

```
void read_and(void *(do_something)(void)) {
    enum read_header status;
    while (...) {
        status = read_header();
        switch (status) {
        case HEADER_SUCCESS: (*do_something)(); continue;
        case HEADER_ZERO_BLOCK:
          if (ignore_zeros_option) continue;
          else break;
        ...
        default: break;
        }
    }
    ...
}

/* Supports all options: -x, -t, -P, -i, ... */
int main(int argc, char **argv) {
    int optchar;
    while (optchar = getopt_long(argc, argv) != -1) {
        switch(optchar) {
        case 'x': read_and(&extract_archive); break;
        case 't': read_and(&list_archive); break;
        case 'P': absolute_names = 1; break;
        case 'i': ignore_zeros_option = 1; break;
        ...
        }
    }
    ...
}
```

Program Debloating

- More code = more vulnerable!
- Aggressively remove unnecessary code so that reduce attack surface
- For example, GNU tar with 97 cmd line options

Global variable declarations removed

Code containing CVE removed

Overwriting functionalities removed

```
int absolute_names;
int ignore_zeros_option;
struct tar_stat_info stat_info;

char *safer_name_suffix (char *file_name, int link_target) {
    int prefix_len;
    char *p;

    if (absolute_names) {
        p = file_name;
    } else {
       /* CVE-2016-6321 */
       /* Incorrect sanitization if “file_name” contains ".." */
       ...
    }
    ...
    return p;
}

void extract_archive() {
    char *file_name = safer_name_suffix(stat_info.file_name, 0);
    /* Overwrite “file_name” if exists */
    ...

void list_archive() { ... }
```

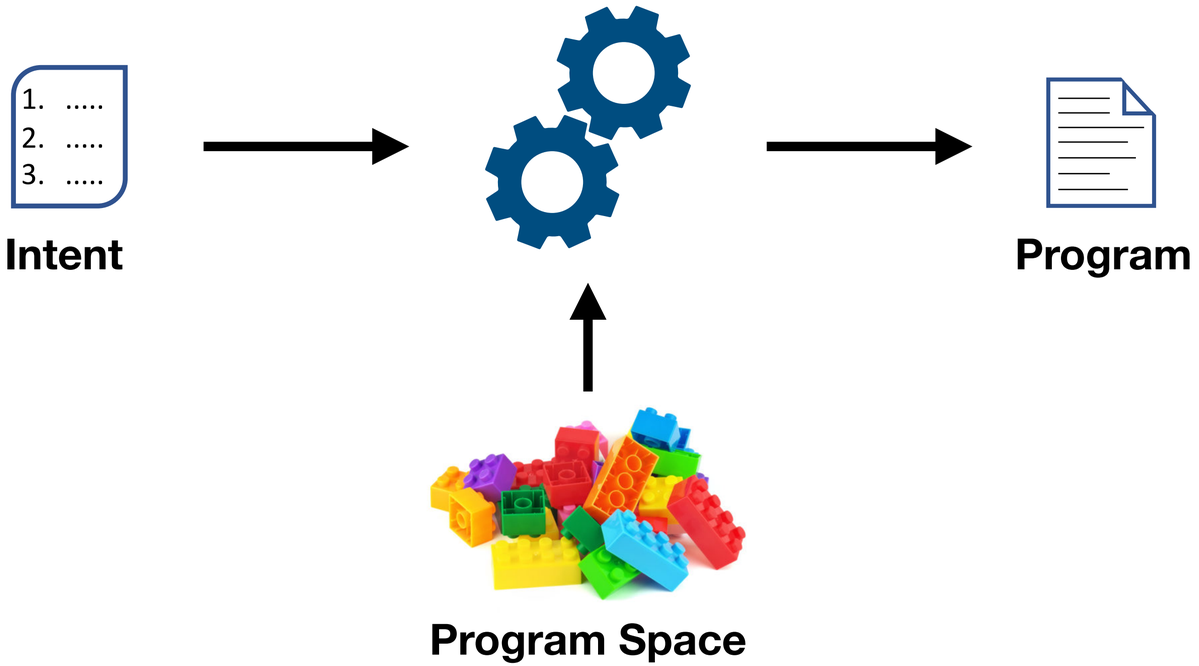
```
void read_and(void *(do_something)(void)) {
    enum read_header status;
    while (...) {
        status = read_header();
        switch (status) {
        case HEADER_SUCCESS: (*do_something)(); continue;
        case HEADER_ZERO_BLOCK:
          if (ignore_zeros_option) continue;
          else break;
        ...
        default: break;
        }
    }
    ...
}

/* Supports all options: -x, -t, -P, -i, ... */
int main(int argc, char **argv) {
    int optchar;
    while (optchar = getopt_long(argc, argv) !=
        switch(optchar) {
        case 'x': read_and(&extract_archive); break;
        case 't': read_and(&list_archive); break;
        case 'P': absolute_names = 1; break;
        case 'i': ignore_zeros_option = 1; break;
        ...
        }
    }
    ...
}
```

Unnecessary functionalities removed

Unsupported options removed

Program Synthesis

- A task of automatically finding a program
  - that satisfy user intent from the underlying program space



History

"Instruction tables will have to be made up by mathematicians with computing experience and perhaps a certain puzzle-solving ability.

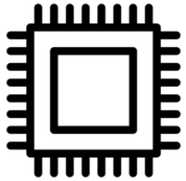
...

There need be no real danger of it ever becoming a drudge, for any processes that are quite mechanical may be turned over to the machine itself."

- A. M. Turing, "Proposed Electronic Calculator", 1946

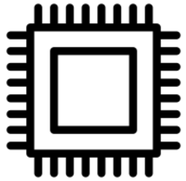
History of Programming

```
quickSort:
.LFB2:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $32, %rsp
        movq    %rdi, -24(%rbp)
        movl    %esi, -28(%rbp)
        movl    %edx, -32(%rbp)
        movl    -28(%rbp), %eax
        cmpl    -32(%rbp), %eax
        jge     .L9
        movl    -32(%rbp), %edx
        movl    -28(%rbp), %ecx
        movq    -24(%rbp), %rax
        movl    %ecx, %esi
        movq    %rax, %rdi
        call    partition
        movl    %eax, -4(%rbp)
        movl    -4(%rbp), %eax
        leal    -1(%rax), %edx
        movl    -28(%rbp), %ecx
        movq    -24(%rbp), %rax
        movl    %ecx, %esi
        movq    %rax, %rdi
        call    quickSort
        movl    -4(%rbp), %eax
        leal    1(%rax), %ecx
        movl    -32(%rbp), %edx
        movq    -24(%rbp), %rax
        movl    %ecx, %esi
        movq    %rax, %rdi
        call    quickSort
.L9:
        nop
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

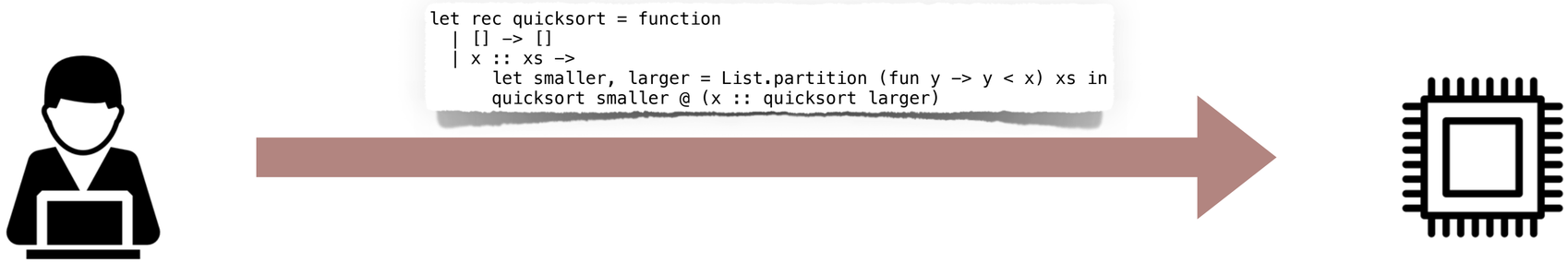


History of Programming

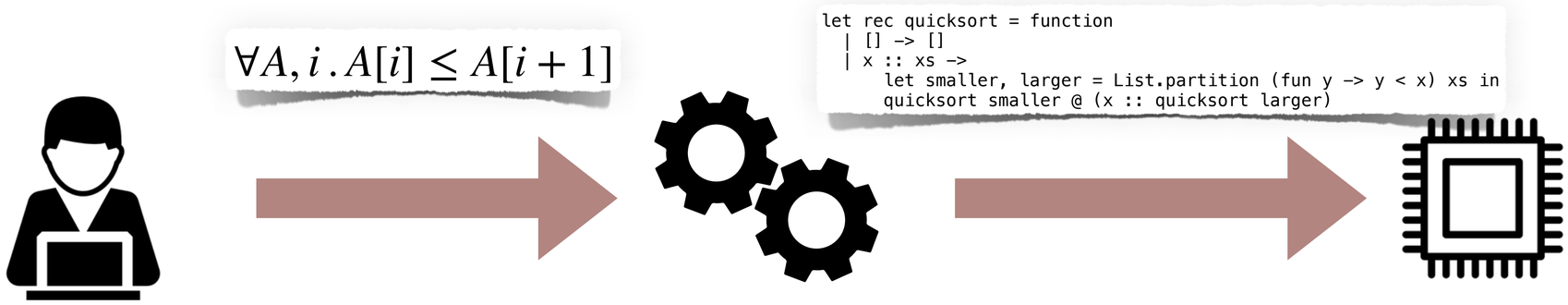
```
void swap(int *a, int *b) {
  int t = *a;
  *a = *b;
  *b = t;
}

int partition(int array[], int low, int high) {
  int pivot = array[high];
  int i = (low - 1);
  for (int j = low; j < high; j++) {
    if (array[j] <= pivot) {
      i++;
      swap(&array[i], &array[j]);
    }
  }
  swap(&array[i + 1], &array[high]);
  return (i + 1);
}

void quickSort(int array[], int low, int high) {
  if (low < high) {
    int pi = partition(array, low, high);
    quickSort(array, low, pi - 1);
    quickSort(array, pi + 1, high);
  }
}
```



History of Programming



Future of Programming



Why Synthesis?

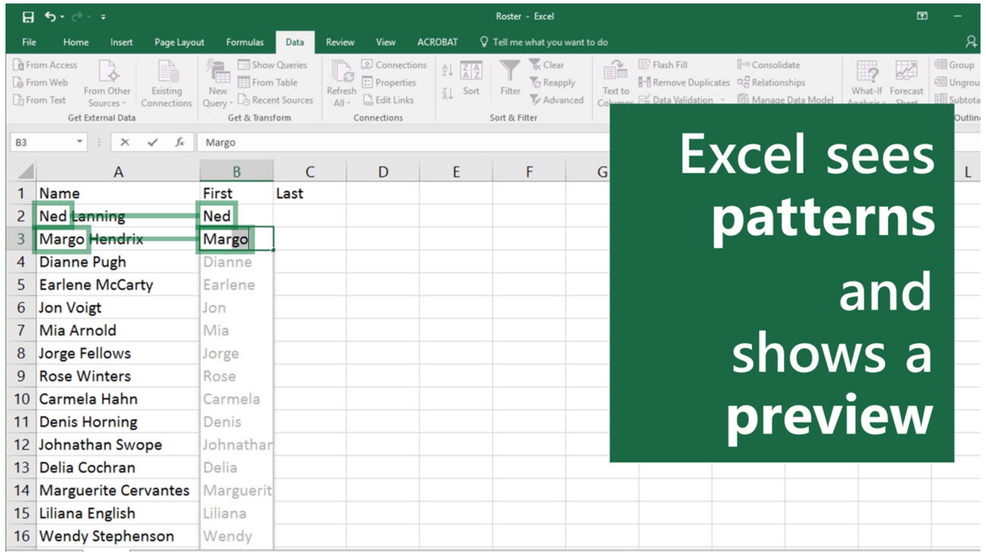
- Human makes mistakes
- #developers << #programming tasks (feature reqs, bug fixes)
  - E.g., Linux kernel's open bug reports
- Programming for everybody

```
  ...
  hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
  hashOut.length = SSL_SHA1_DIGEST_LEN;
  if ((err = SSLFreeBuffer(&hashCtx)) != 0)
      goto fail;
  if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
      goto fail;
  if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
      goto fail;
  if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
      goto fail;
  if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
      goto fail;
      goto fail;  /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
  if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
      goto fail;

  err = sslRawVerify(…);
  ...
fail:
  ...
  return err;
```

goto fail, 2014
MacOS / iOS
CVE-2014-1266

Application: End-user Programming (1)

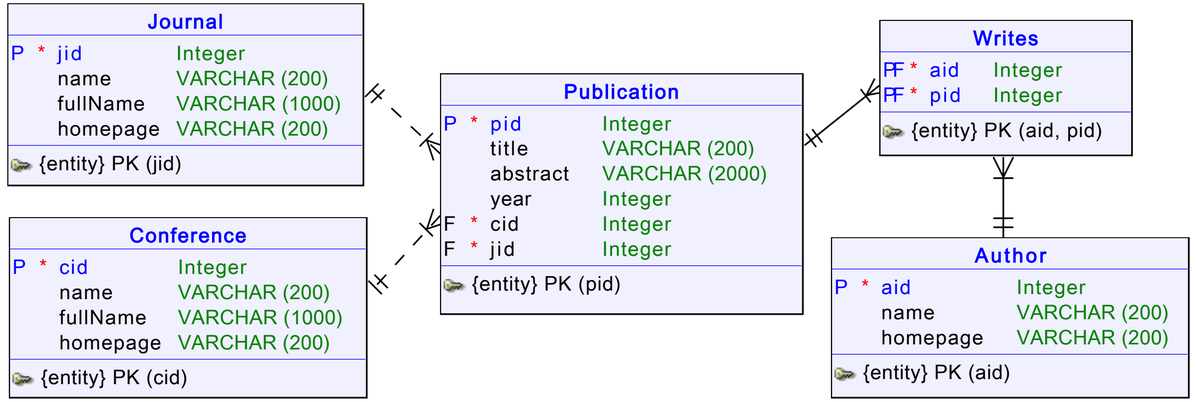
- String transformation (e.g., MS Excel's FlashFill)



Automating String Processing in Spreadsheets using Input-Output Examples, POPL'11

Application: End-user Programming (2)

- SQL query synthesis



"Find the number of papers in OOPSLA 2020"

```
SELECT count(Publication.pid)
FROM Publication JOIN Conference ON Publication.cid = Conference.cid
WHERE Conference.name = "OOPSLA" AND Publication.year = 2010
```

SQLizer: Query Synthesis from Natural Language, OOPSLA'17

Challenges

시작하기 × int check_buffer_overflow() { Untitled-2

```c
int check_buffer_overflow() {
    int buffer[5];
    buffer[6] = 0;
    return 0;
}
```

```c
// read an image file
int read_file() {
    FILE *fp;
    int i, j, k;
    int n;
    int m;
    int nc;
    int nr;
    int nb;
    int nc_max;
    int nr_max;
    int nb_max;
    int nc_min;
    int nr_min;
    int nb_min;
    int nc_avg;
    int nr_avg;
    int nb_avg;
    int nc_sum;
    int nr_sum;
    int nb_sum;
    int nc_var;
    int nr_var;
    int nb_var;
    int nc_std;
    int nr_std;
    int nb_std;
    int nc_med;
    int nr_med;
    int nb_med;
    int nc_mode;
    int nr_mode;
    int nb_mode;
    int nc_min_index;
    int nr_min_index;
    int nb_min_index;
    int nc_max_index;
    int nr_max_index;
    int nb_max_index;
    int nc_med_index;
    int nr_med_index;
    int nb_med_index;
    int nc_mode_index;
    int nr_mode_index;
    int nb_mode_index;
    int nc_sum_index;
    int nr_sum_index;
    int nb_sum_index;
    int nc_var_index;
    int nr_var_index;
    int nb_var_index;
    int nc_std_index;
    int nr_std_index;
    int nb_std_index;
    int nc_avg_index;
    int nr_avg_index;
    int nb_avg_index;
    int nc_min_index_index;
    int nr_min
}
```

Summary

- Error-free software: Holy grail of CS

CS492: Program Reasoning

- Automatically verify the correctness of a program: undecidable
- Practical solution: either sound or complete (or none of them)
  - testing: unsound but complete
  - program analysis: sound but not complete

CS524: Program Reasoning

- Emerging technologies
  - program debloating: why not remove unused code?
  - program synthesis: why not automatically generate code?

CS492: Program Reasoning