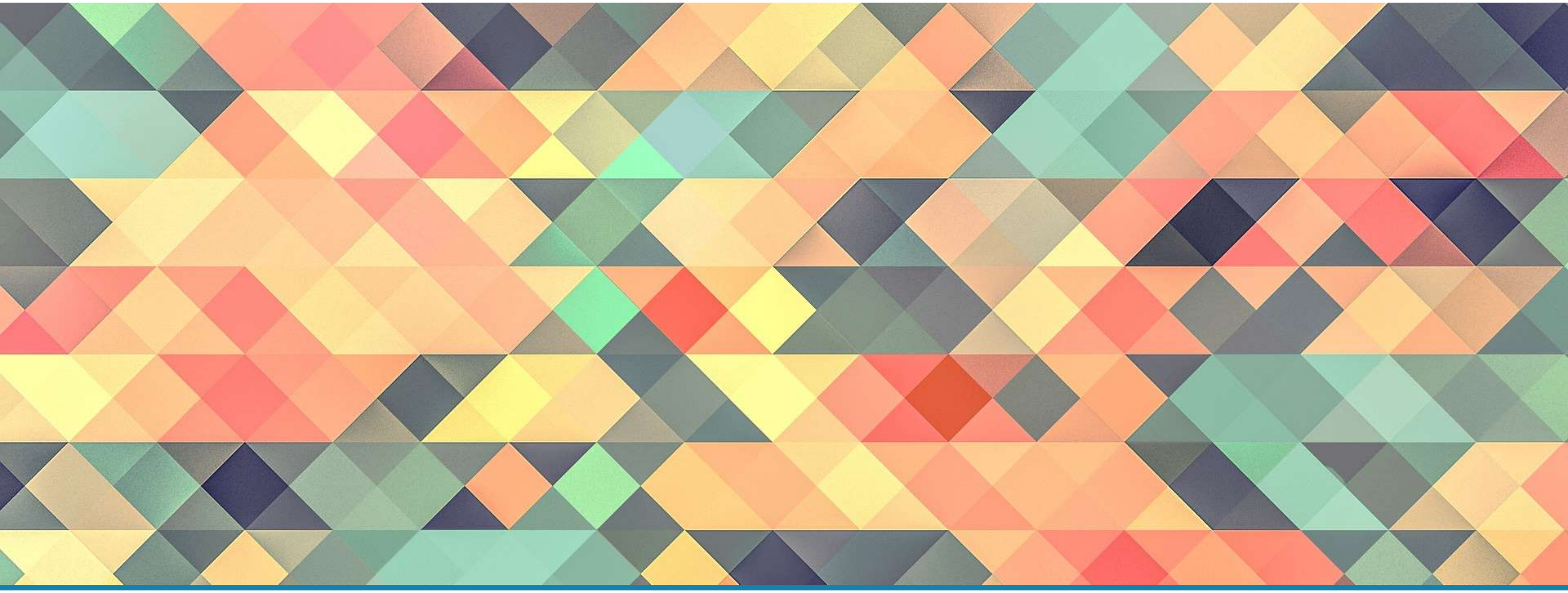


SOFTWARE CORRECTNESS TOOLS

Mark Gritter
Minnebar 2019

ABOUT ME

Twitter: @markgritter

Github: mgritter

ex-Tintri (co-founder and lead architect)

Vault Advisor at Hashicorp

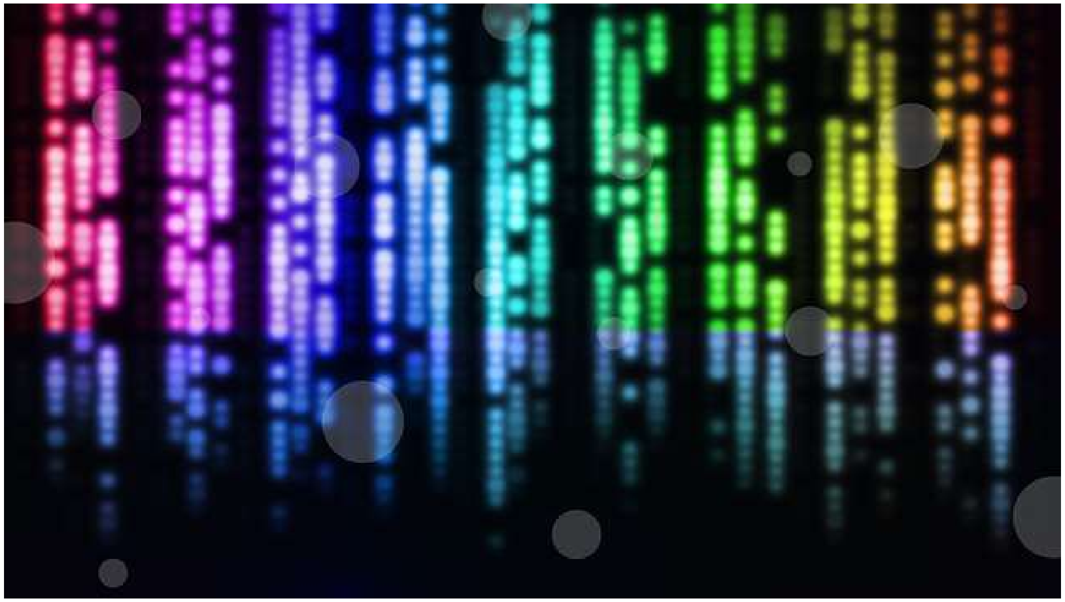
This talk does not reflect the opinions of my current or former employers.

SOFTWARE SUCKS

Software is hard.

"Try very hard and think a lot" is a failed solution.

Even very smart people miss simple edge cases.



WHAT DO SOFTWARE CORRECTNESS TOOLS DO?

Static analysis

Dynamic analysis

Analyze the structure

Measure the behavior

Find novel inputs

Linter

Proof assistant

Type checking

Memory checker

Virtual machine

Visualization

Test case generator

Model checker

Symbolic execution

Fuzzer

SOFTWARE CORRECTNESS TOOLS AND DEVOPS

Analyze the structure

Measure the behavior

Find novel inputs

Code review

Monitoring / observability

Benchmarks

Coverage metrics

Test plans

Chaos Engineering

STRENGTHS AND WEAKNESSES

A software correctness tool has value because it has a *different* set of biases than human programmers and testers.

Methodical, but dumb.

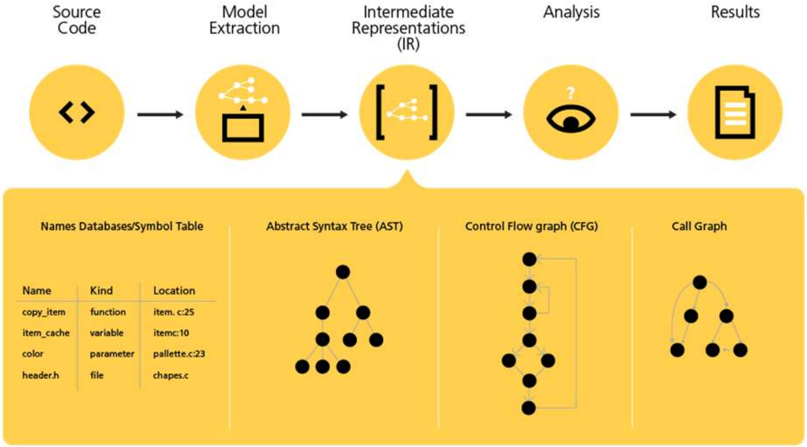
Humans come in with an understanding of how code is "supposed to work"; tools do not.

Humans are great at experimenting and exploring… for about 20 tests, then they get bored.

STATIC ANALYSIS

Tons of open source and commercial tools:

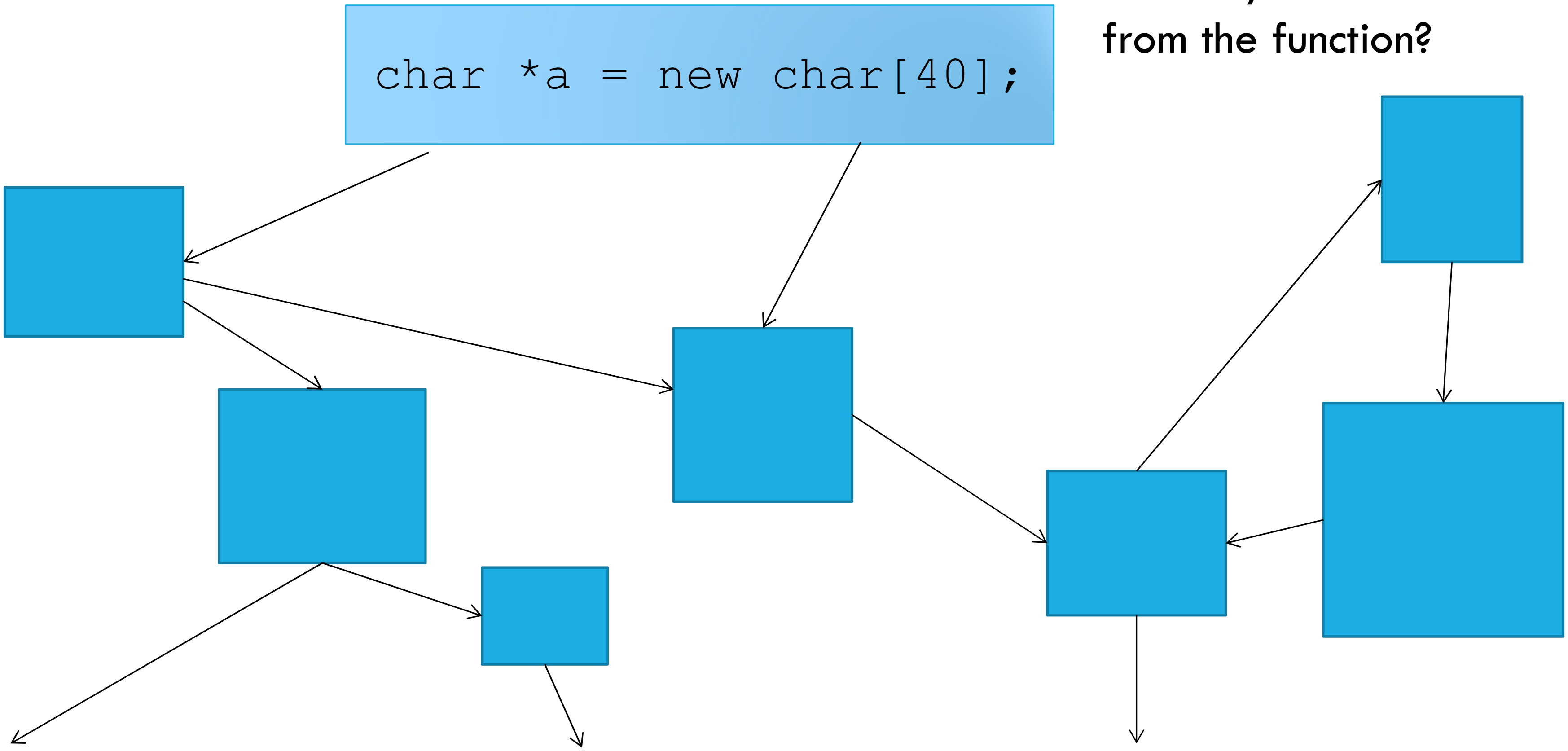
- C and C++: Coverity, cpplint
- C# and .NET: StyleCop, Parasoft
- Java: FindBugs/SpotBugs,
- JS: JSLint/JSHint
- Python: Pylint, PyCharm
- As a service: SonarQube, Veracode


Source Code
Model Extraction
Intermediate Representations (IR)
Analysis
Results
Names Databases/Symbol Table
Abstract Syntax Tree (AST)
Control Flow graph (CFG)
Call Graph

Name	Kind	Location
copy_item	function	item. c:25
item_cache	variable	itemc:10
color	parameter	pallette.c:23
header.h	file	chapes.c

STATIC ANALYSIS EXAMPLE

Is this memory stored or freed on every exit from the function?



```
char *a = new char[40];
```

STATIC ANALYSIS DOWNSIDES

If you haven’t been using one from day 1, you usually have an immense backlog.

Little support for honing in on which patterns are meaningful errors.

Does it work with your compiler?

Developers argue with the tools: “that’s not really a bug.”

FUZZERS

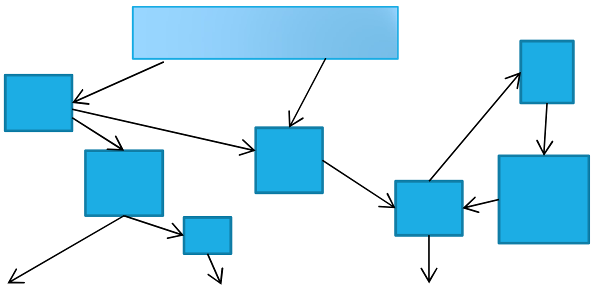
Send random data to a program and see what happens!

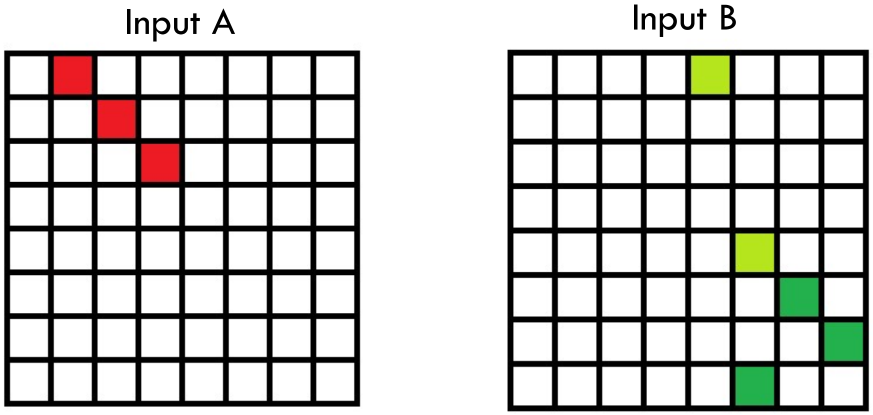
(You don’t need a tool to do this, you can do it as part of unit or integration testing.)

ADVANCED FUZZING

Instrument the code (compile-time or run-time) to check code (or data!) coverage from inputs.

Prioritize variations of inputs that lead to new patterns of execution.





EXAMPLE FUZZERS

AFL (American Fuzzy Lop): demonstrated the power of coverage-based fuzzing

libFuzzer: included as part of clang/LLVM

go-fuzz

Syzcaller: Linux kernel syscall fuzzing, run continuously

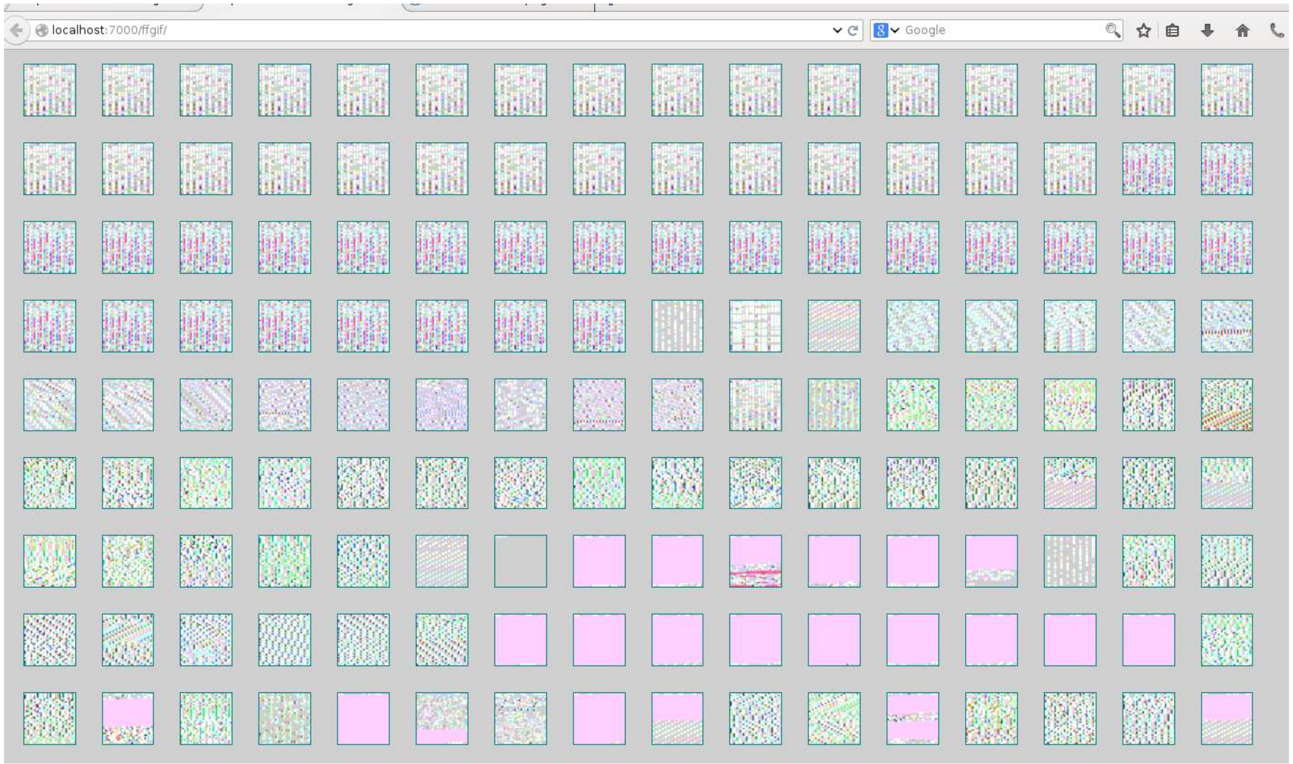
All of these have *mountains* of bugs and CVE's to their credit.

- If you've never run a fuzzer against a parser, it's likely it will find bugs
- I did this both at Tintri and as a side project

AN EXAMPLE BUG

Firefox leaked uninitialized memory when asked to display malformed GIFs.

- https://bugzilla.mozilla.org/show_bug.cgi?id=1045977
- CVE-2014-1564
- In this case, fuzzing combined with memory checking showed the error.



FUZZING DOWNSIDES

Really good for finding crashes and vulnerabilities in parsers, less good for checking program logic or memory leaks.

Generally need to either already take input from standard input, or write a harness.

- If you wrote a randomized test, consider taking its input from a fuzzer instead.

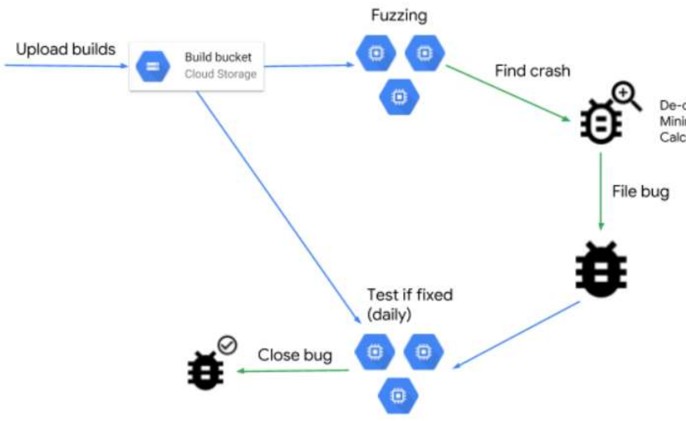
Sometimes requires program tweaks to work well, for example disabling checkums. Dictionary-based approaches can help with special keywords, or specialized input generators like ProtoFuzz

FUZZING AS-A-SERVICE

Google OSS-Fuzz (and ClusterFuzz)

https://github.com/google/oss-fuzz

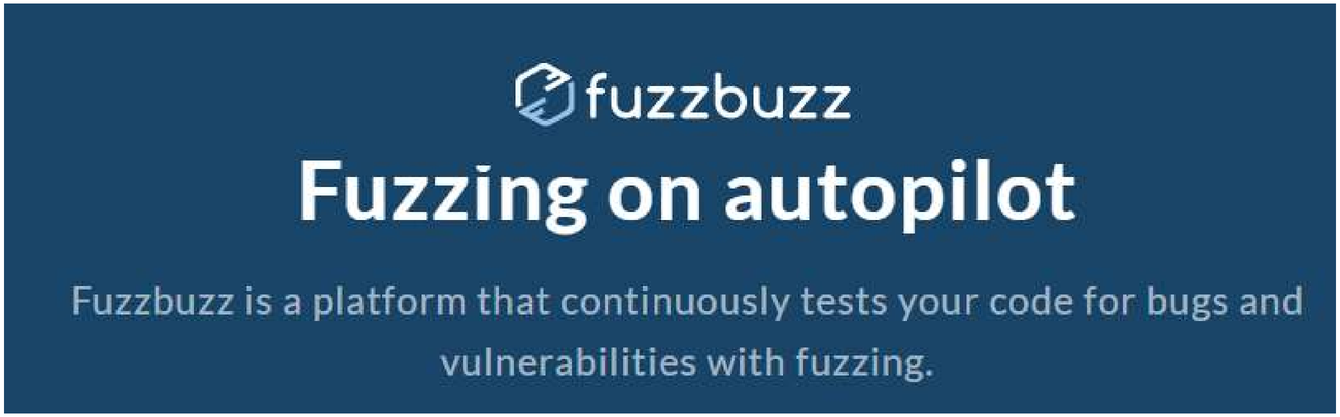
Today, we're announcing that ClusterFuzz is now open source and available for anyone to use.



Microsoft Security Risk Detection

Scalable fuzz lab in the cloud

One click scalable, automated, Intelligent Security testing lab in the cloud.



fuzzbuzz

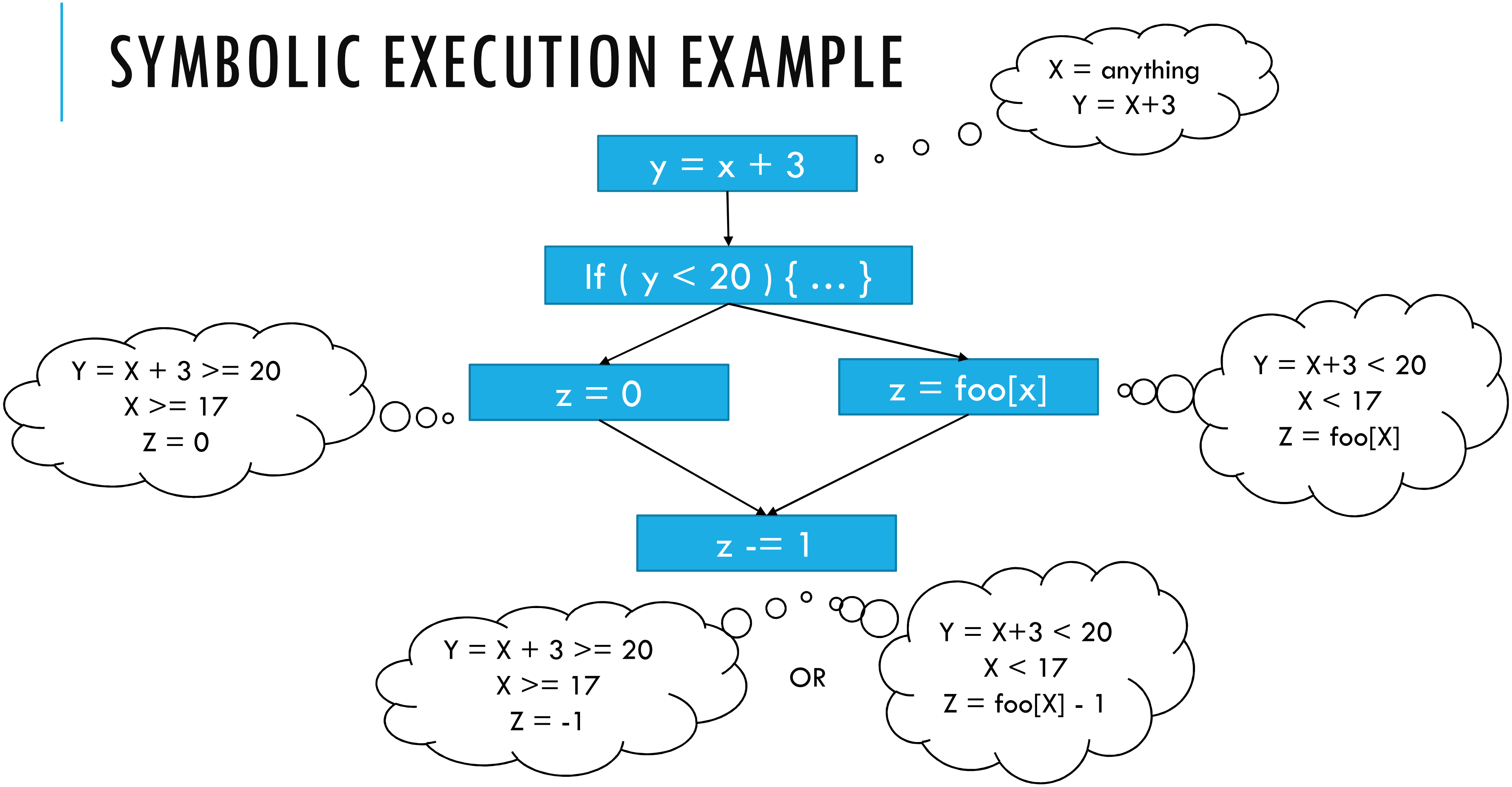
Fuzzing on autopilot

Fuzzbuzz is a platform that continuously tests your code for bugs and vulnerabilities with fuzzing.

SYMBOLIC EXECUTION

Replace inputs with unbound variables, and use algebra and logic.

SYMBOLIC EXECUTION EXAMPLE



X = anything
Y = X+3

y = x + 3

If (y < 20) { … }

Y = X + 3 >= 20
X >= 17
Z = 0

z = 0

z = foo[x]

Y = X+3 < 20
X < 17
Z = foo[X]

z -= 1

Y = X + 3 >= 20
X >= 17
Z = -1

OR

Y = X+3 < 20
X < 17
Z = foo[X] - 1

SYMBOLIC EXECUTION SYSTEMS

KLEE (LLVM), KeY (Java), Otter (C), SymJS (Javascript), Mayhem (commercial from ForAllSecure, various)

Often based on SMT (Satisfiability Modulo Theory) provers

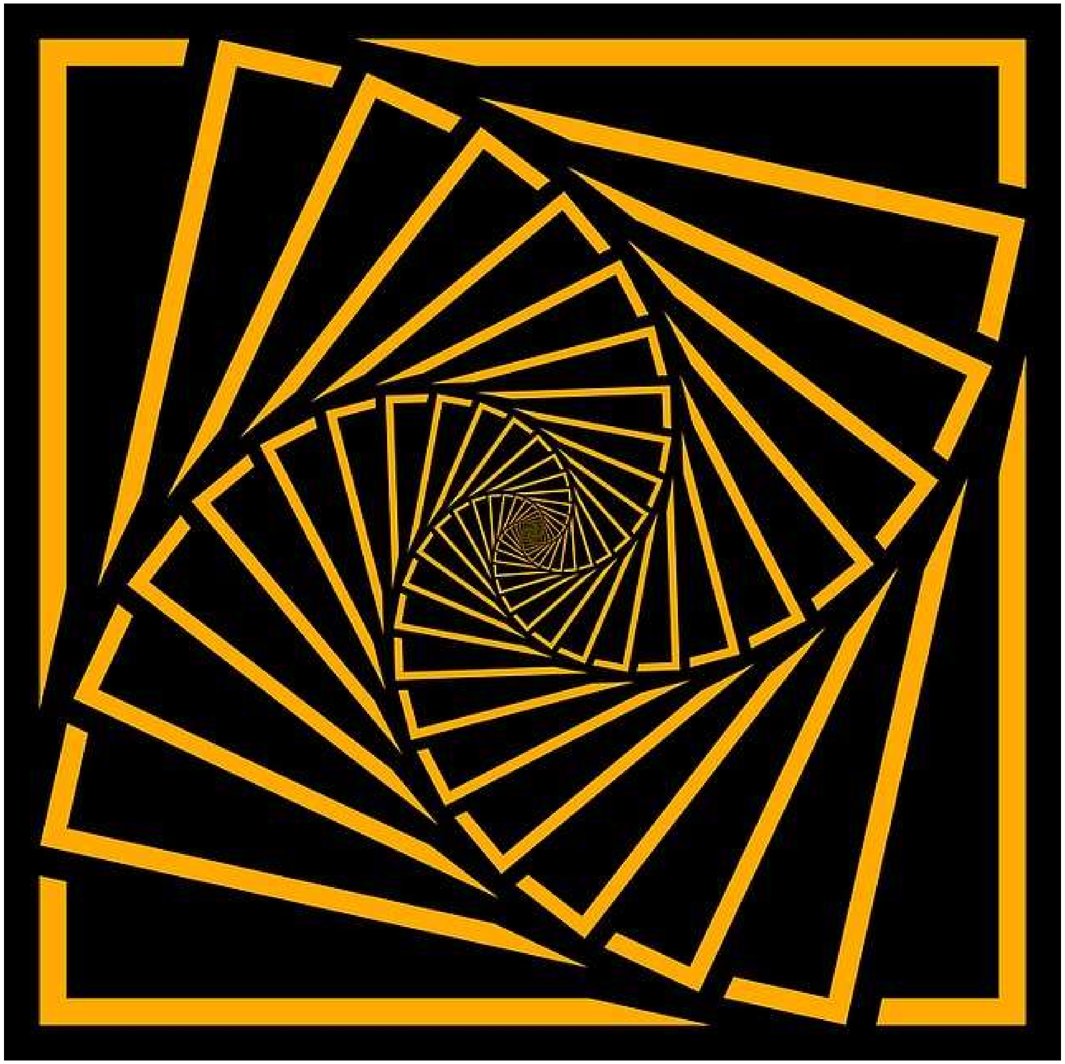
- Boolean satisfiability along with integer arithmetic mod 2^N
- Examples: Z3, STP, Yices

Computationally expensive, but getting better.

VIRTUAL MACHINES

Augment the "usual" memory model with additional state.

Usually slower, but can run real programs and find memory errors and race conditions.

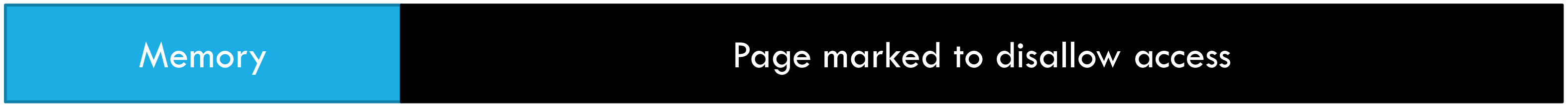


MEMORY CHECKING

Popular examples: Valgrind, MemorySanitizer

Methods:

- White-box: Rewrite every read and write to bounds-check
- Black-box: Use operating system memory protection to put guards around allocated memory



Memory

Page marked to disallow access

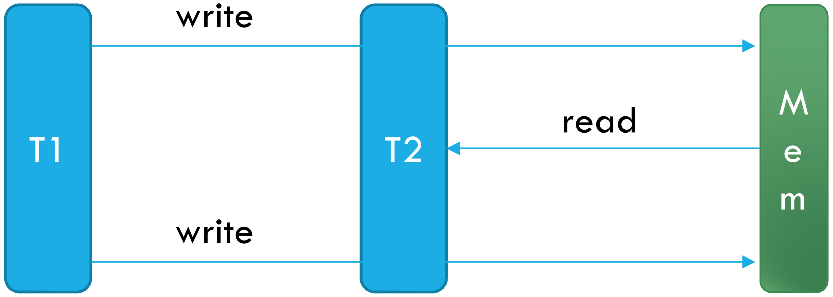
- Black-box: Keep backtrace of each memory allocation, show which locations allocate leaked memory.

CONCURRENCY CHECKERS

ThreadSantizer, used in Go race detector, Parallocity ZVM (defunct)

When memory is read or written:

- Record which thread, and which locks were held
- If subsequent accesses from a different thread don't use a lock, it's a race condition.



DATA TAINTING

A recent example from Google Project Zero: “Detecting Kernel Memory Disclosure with x86 Emulation and Taint Tracking”

- Run the kernel in Bochs
- Track “taint” on new stack, heap, or pool allocations
- Propagate the taint to copies
- Erase taint when memory is overwritten with constant values

Found 65 CVEs in Windows!

RUNTIME CHECKING PRO/CON

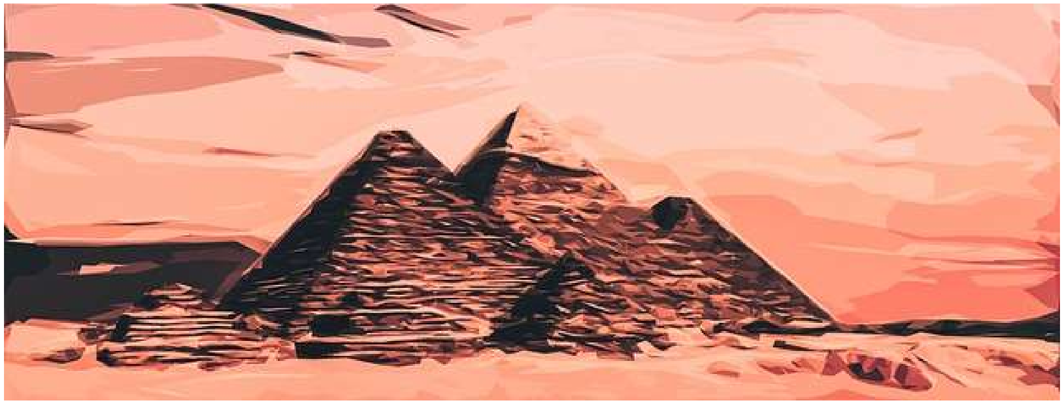
Can usually be run on even the most complicated software, but very memory-hungry. Research in this area is usually about how to use less memory.

Some amount of false positives, particularly in concurrency checking. Nonstandard usage can't be checked like application-specific memory allocation or locking.

Probably need to combine with another technique to get good coverage, but very useful if you already know the general area to look.

MODEL CHECKERS

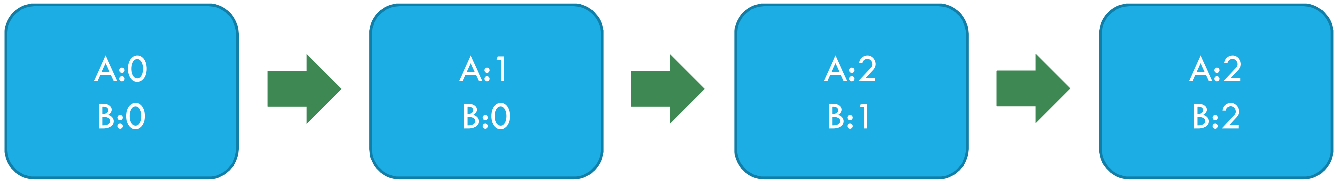
Explore whether an invariant holds on a given description of the software.



TLA+

Written by Leslie Lamport, “Temporal Logic of Actions”

Describes the logical relationship between steps of the computation (or other process!) using the language of set theory.



A:0
B:0

A:1
B:0

A:2
B:1

A:2
B:2

TLC model checker enumerates all possible “behaviors” obeying this specification and checks invariants, or deadlock.

Used by Amazon to find bugs in AWS. I reproduced a simple concurrency bug from work. Good for concurrent systems.

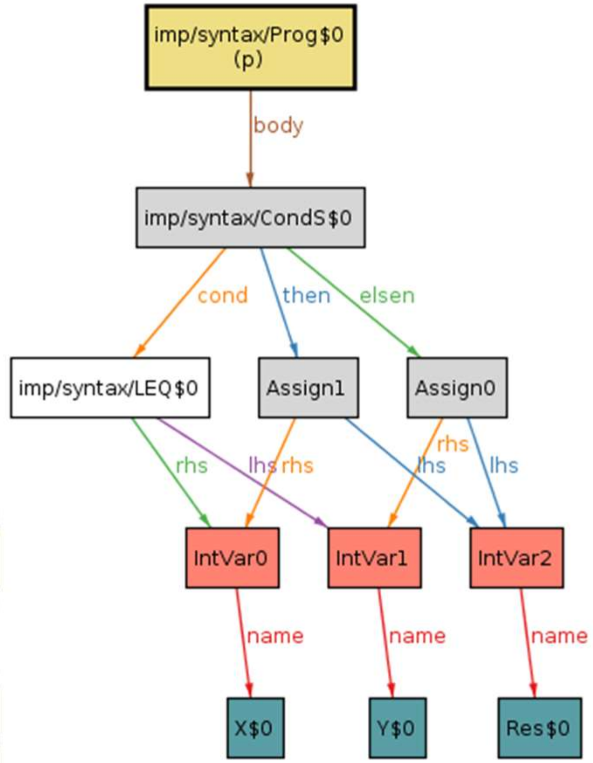
Hillel Wayne (@hillelogram) has a book out this year “Practical TLA+”

ALLOY

Relational model that uses a SAT solver to check invariants, or prove an example exists.

Good integration with graphing tool to show examples; easier to get started than TLA+, but somewhat more difficult to represent concurrency. Great for data structures.

Pamela Zave used Alloy to find bugs in the distributed hash table Chord.



```
// Move x to directory d
pred move [fs, fs': FileSystem, x: FSObject, d: Dir] {
  (x + d) in fs.live
  fs'.parent = fs.parent - x->(x.(fs.parent)) + x->d
}
```

MODEL CHECKING LIMITATIONS

You can’t tell how well your design actually represents the code that got written; are you updating it as the code changes?

Exhaustive testing of models is expensive; a lot of the “folk knowledge” is how to represent a system in a feasible way.

But, this is great for finding a different class of bugs than techniques that only work on code.

Other model checkers: SPIN/Promela, BLAST, PRISM

PROOF ASSISTANTS

Verify that the stated properties of a program hold, using hints from the programmer.

- Programmer will often have to write a loop invariant or other property that can then be proved or checked.
- Interactive process: tool will try to indicate where it got lost.

A lot of overlap with purely mathematical theorems; many tools can be used for both.

- Examples: Isabelle, CoQ

KEY

KeY (key-project.org) incorporates a lot of the ideas:

- Annotate Java program in a modelling language
- Prove the assertions specified in the modelling language
- Rule-based symbolic execution
- SMT solver

Works with real Java code; unfortunately, no floating point or Generics or Java 8 features.

KEY'S MOST FAMOUS BUG, TIMSORT

```
/*@ private normal_behavior
  @ requires
  @   n >= MIN_MERGE;
  @ ensures
  @   \result >= MIN_MERGE/2;
  @*/

private static int /*@ pure @*/ minRunLength(int n) {
  assert n >= 0;
  int r = 0;      // Becomes 1 if any 1 bits are shifted off
  /*@ loop_invariant n >= MIN_MERGE/2 && r >=0 && r<=1;
    @ decreases n;
    @ assignable \nothing;
    @*/
  while (n >= MIN_MERGE) {
    r |= (n & 1);
    n >>= 1;
  }
  return n + r;
}
```

see my Minnebar 2015 talk,
"Never Trust Any Published Algorithm"
https://www.youtube.com/watch?v=S2uEUghfiXY

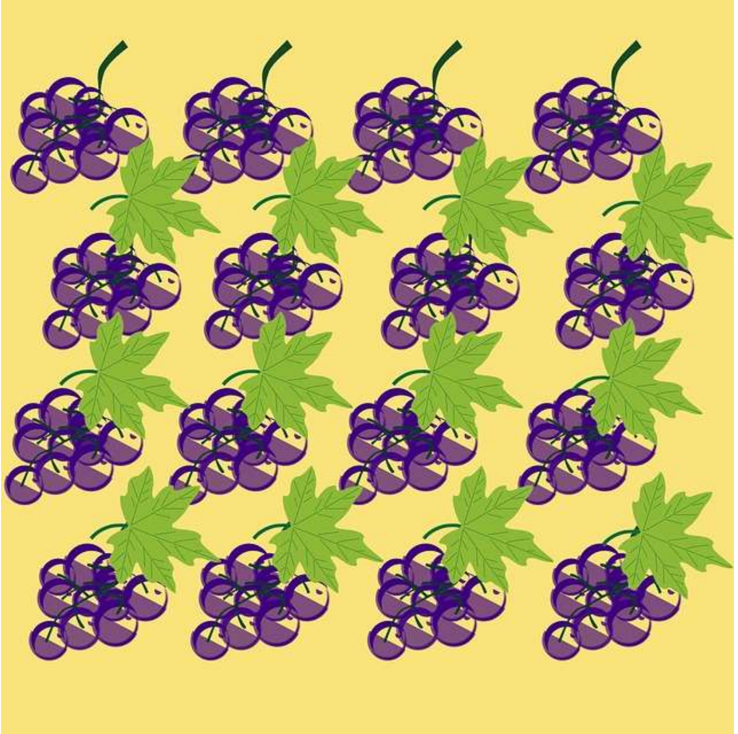
PROPERTY-BASED TESTING

Generate inputs from a schema and validate the results (using an alternate implementation, or an assertion.)

"For all x, if P(x) holds, does Q(x) hold?"

Examples:

- QuickCheck (Haskell)
- fast-check (Javascript)
- Hypothesis (Python)
- RapidCheck (C++)

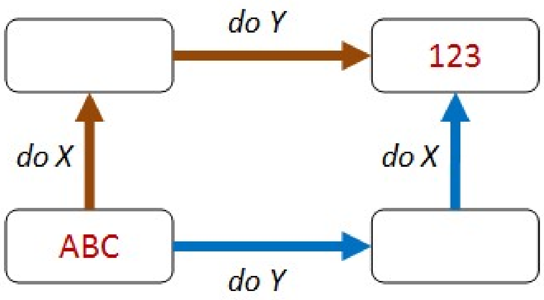


HYPOTHESIS EXAMPLE

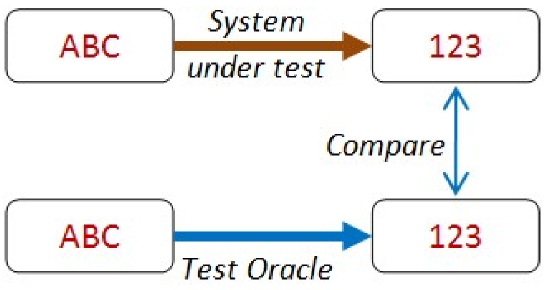
```
from hypothesis import given
from hypothesis.strategies import text

@given(text())
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

WAYS TO PROPERTY TEST

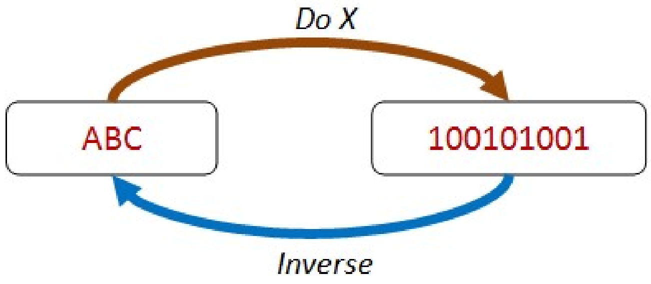
commutative



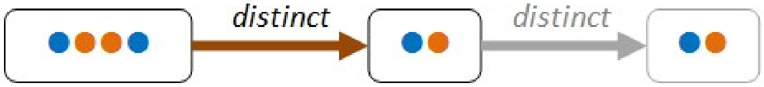
oracle



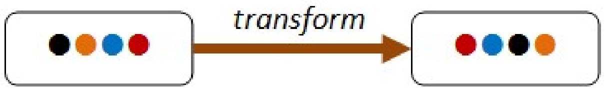
inverse operation



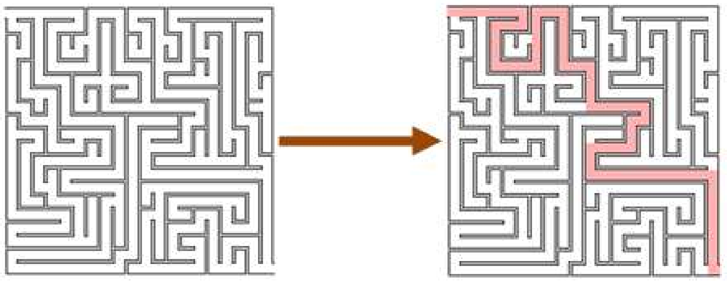
idempotent



invariant



verification



Examples from: https://fsharpforfunandprofit.com/posts/property-based-testing-2/

PROPERTY TESTING PRO/CON

If you are writing unit tests, this is almost a freebie. Writing meaningful test case generation can be a little challenging.

- But if you're writing randomized tests already, consider using a property-based framework to get test case minimization

Runs quickly, but coverage can be hit-or-miss; most property-based testers don't yet take advantage of coverage information.

TEST GENERATION

Test case generation: Given a model or an implementation, generate test cases

Edge case generation: Heuristically pick inputs that are likely to trigger edge cases.

Test case minimization: Find test inputs that cover the space of behaviors, or are the smallest example of a particular behavior.

Examples: Randoop, Hypothesis, Tcases, AgitarOne, Jtest, Evosuite, UniqueSoft D*Code, Conformiq

- Often commercial tools are aimed at generating regression tests for a legacy piece of software.

NOT COVERED IN THIS TALK

Visualization tools

Observability tools

Load generators

Mocking frameworks

UML

DO BUGS MATTER? LESSONS FROM COVERITY

Companies buy bug-finding tools because they see bugs as bad. **However, not everyone agrees that bugs matter.** The following event has occurred during numerous trials. The tool finds a clear, ugly error (memory corruption or use-after-free) in important code, and the interaction with the customer goes like thus:

"So?"

"Isn't that bad? What happens if you hit it?"

"Oh, it'll crash. We'll get a call." [Shrug.]

If developers don't feel pain, they often don't care. Indifference can arise from lack of accountability; if QA cannot reproduce a bug, then there is no blame.

-- Bessey et al, “A Few Billion Lines of Code Later”, Communications of the ACM, February 2010

CHALLENGE

Think about what tools you could add to augment the things you're already doing!

- *Design reviews:* TLA+, Alloy
- *Code reviews:* Static checker
- *Unit testing:* Hypothesis, libFuzzer
- *Integration testing:* valgrind, MemorySanitizer, Go Race Detector
- *Security audits*: all of the above + ???

All of these are easy to get started with; pick just one small thing to get started and demonstrate the value. You can also use them on bugs you already know about to figure out how you *could* have found them.

THANK YOU!

Twitter: @markgritter

Email: mgritter@gmail.com

Github (w/ slides): mgritter

Side project: @fuzz-ai

(on hiatus)

Images by Denis Azarenko from Pixabay